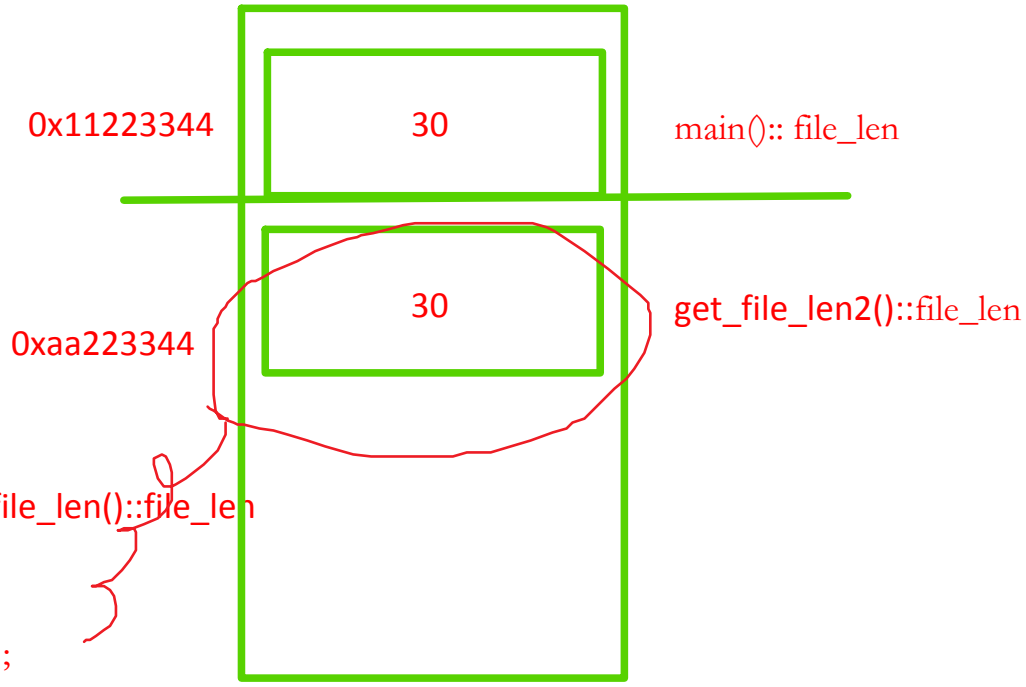
1 间接操作1

2015年11月14日 8:41

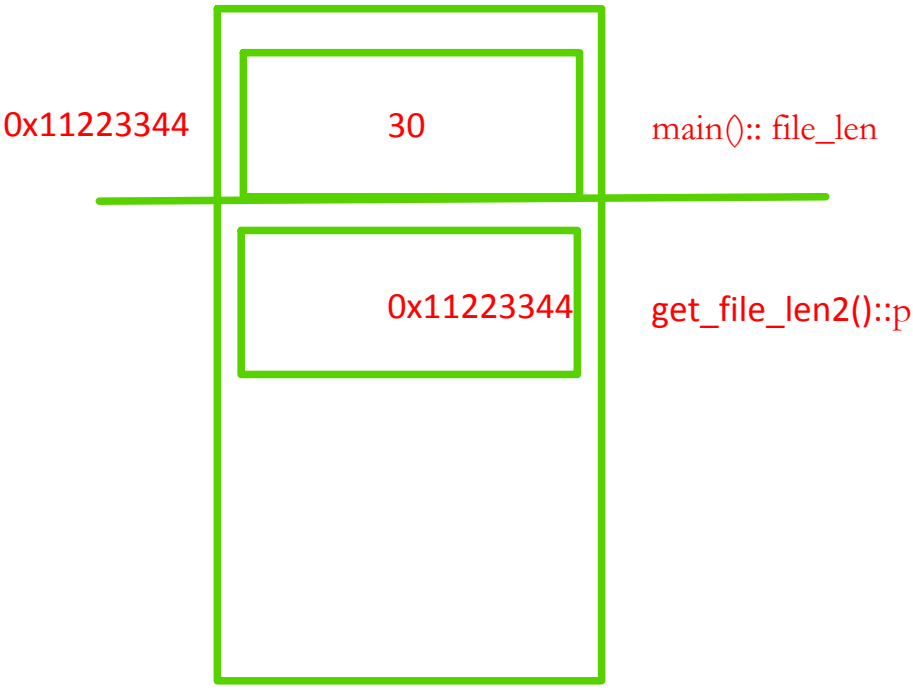
```
int get_file_len2()
{
    int file_len = 0;

    file_len = 30;

    return file_len;
} main::file_len = get_file_len()::file_len
  a = b

file_len = get_file_len2();
```



2 间接操作2

2015年11月14日 9:36

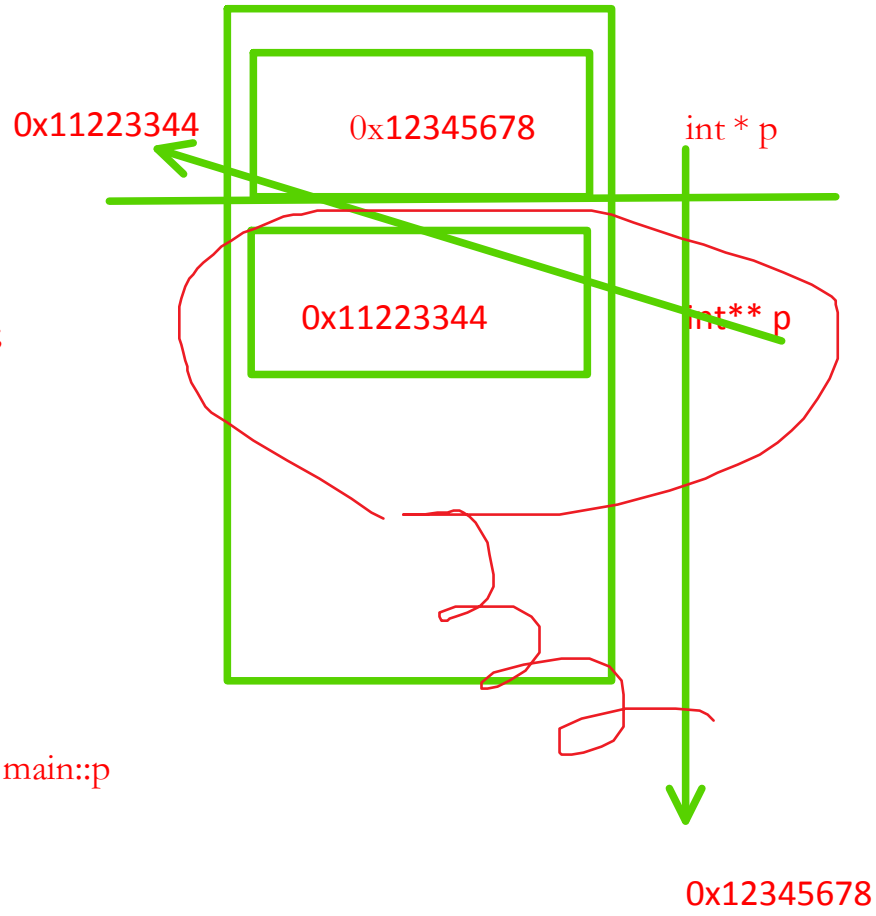
```
int get_file_len3(int *p)
{
        *p = 30;

        return 0;
}
```




```
file_len = get_file_len2();
```

3 间接操作3

2015年11月14日 9:42

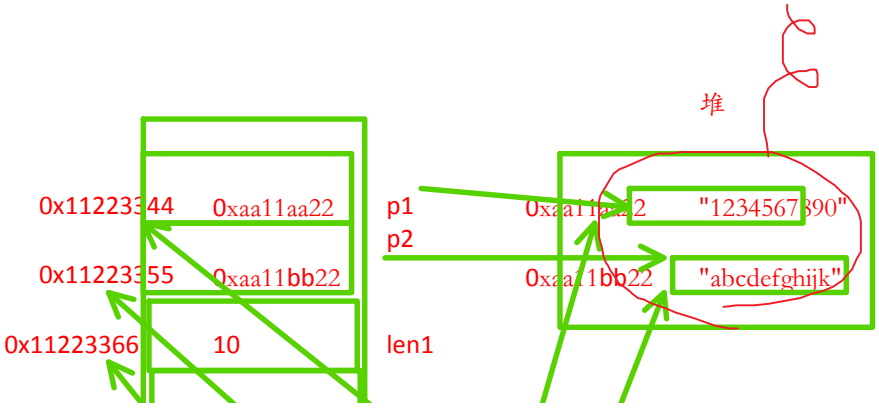



```
void change_pointer(int **p)
{
	*p = (int*)0x12345678;
}

int main(void)
{
	int * p = NULL;

	change_pointer(&p);//
	change_pointer::p = &  main::p

	return 0;
}
```

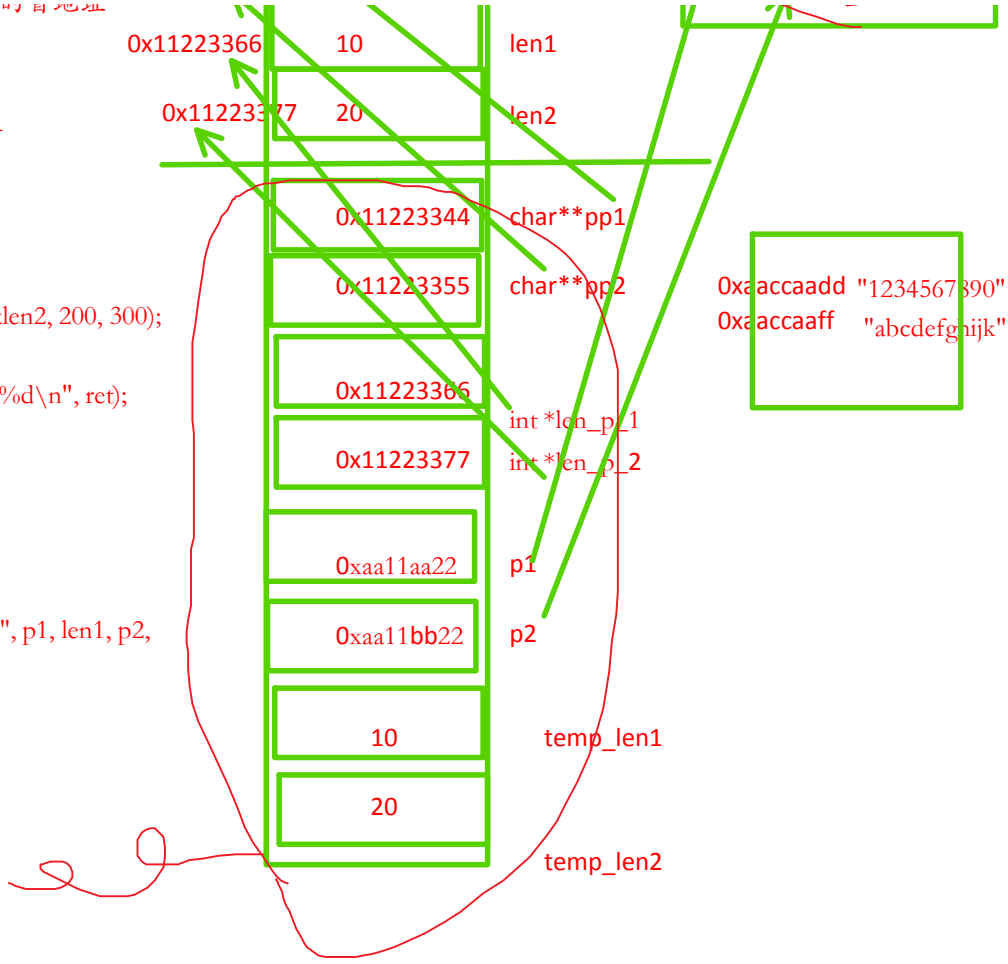
4 开辟内存的四区图

2015年11月14日　　10:19

```
int get_mem(char **pp1, char **pp2, int *len_p_1, int
*len_p_2, int len1, int len2)
{
	char *p1 = NULL;
	*pp1 = malloc(
	char *p2 = NULL;
	int temp_len1 = 0;
	int temp_len2 = 0;


	p1 = (char*)malloc(len1);//p1
	if (p1 == NULL) {
		fprintf(stderr, "malloc p1 err\n");
		return -1;
	}
	memset(p1, 0, len1);


	p2 = malloc(len2);
	if (p2 == NULL) {
		fprintf(stderr, "malloc p2 err\n");
		return -1;
	}
	memset(p2, 0, len2);


	strcpy(p1, "1234567890");
	temp_len1 = strlen(p1);

	strcpy(p2, "abcdefghijk");
	temp_len2 = strlen(p2);



	//到此 以上说明程序没问题了
	*pp1 = p1;
	*pp2 = p2;
	*len_p_1 = temp_len1;
	*len_p_2 = temp_len2; //间接操作

	return 0;
}

int main(void)
{
	c
	char *p1 = NULL; //第一块内存的首地址
	char *p2 = NULL;
```



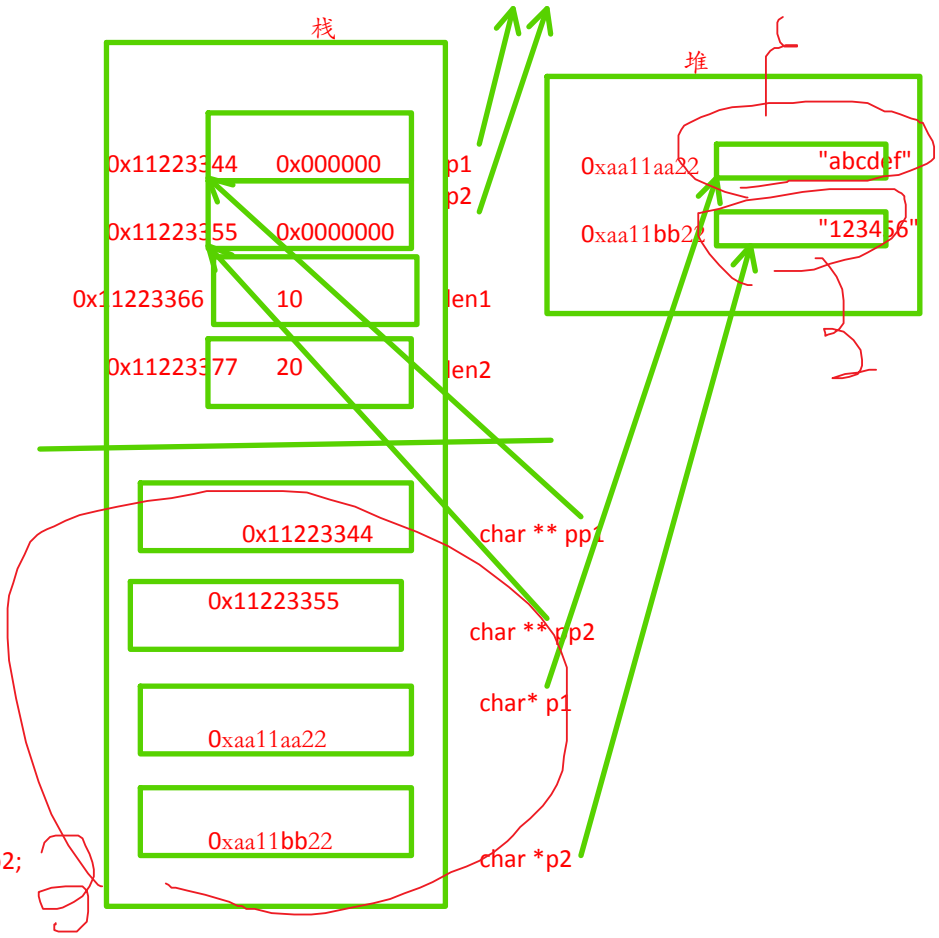
```
char *p2 = NULL;

int len1 = 0; //第一块内存的长度
int len2 = 0;

int ret = 0;

ret = get_mem(&p1, &p2, &len1, &len2, 200, 300);
if (ret != 0) {
        fprintf(stderr, "get mem error: %d\n", ret);
        return -1;
}


//对内存的操作
printf("p1 : %s[%d], p2 : %s[%d]\n", p1, len1, p2,
len2);



if (p1 != NULL) {
        free(p1);
        p1 = NULL;
}

if (p2 != NULL) {
        free(p2);
        p2 = NULL;
}
printf("main4  succ\n");


return 0;
}
```



5 mem_free2

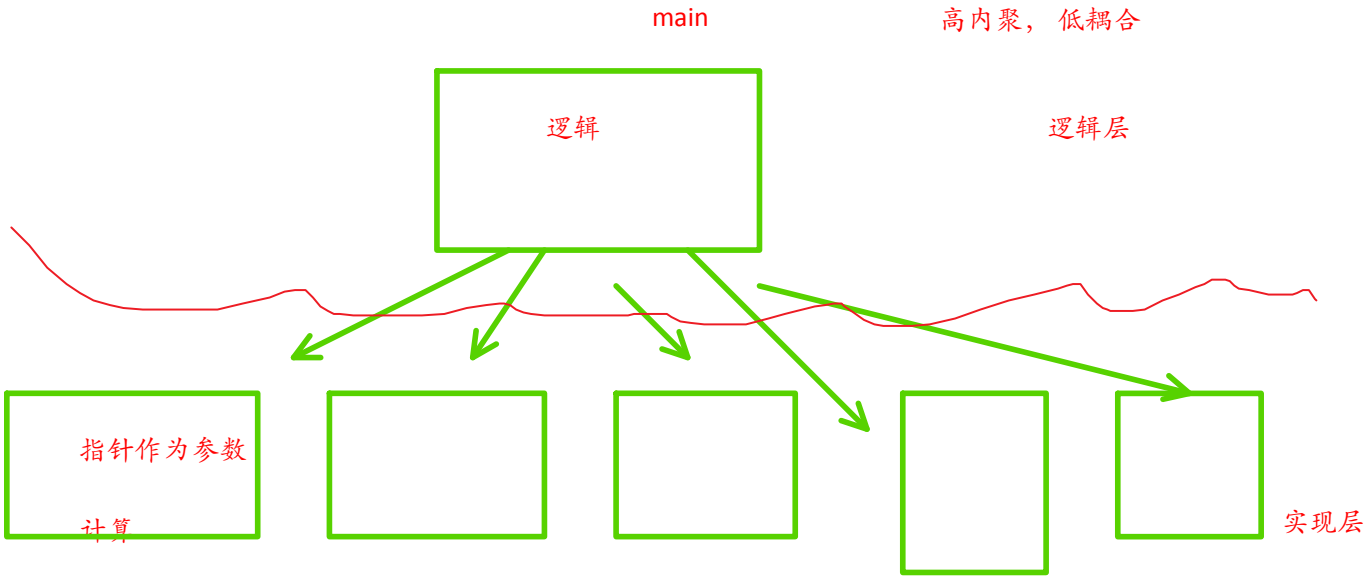
2015年11月14日 10:44

```
void free_mem2(char **pp1, char **pp2)
{
	char *p1 = NULL;
	char *p2 = NULL;

	p1 = *pp1;
	p2 = *pp2;

	if (p1 != NULL) {
		free(p1);
	}
	if (p2 != NULL) {
		free(p2);
	}

	*pp1 = NULL;
	*pp2 = NULL;

}

free_mem2(&p1, &p2); // pp1 = &p1, pp2 = &p2;
```



6 指针作为函数参数的意义

2015年11月14日 10:59

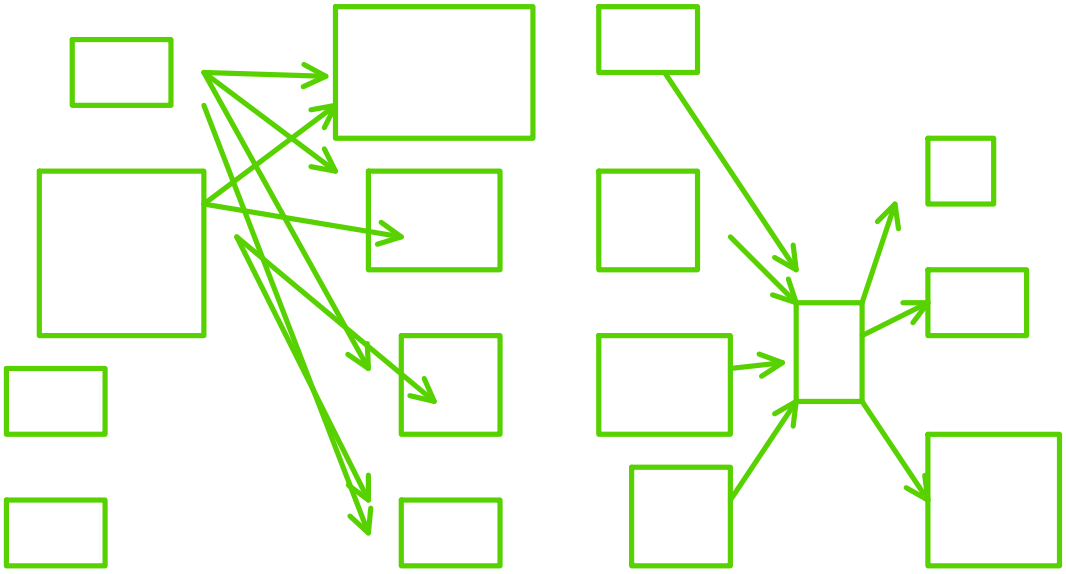


通过指针的穿着，可以在子模块中将已经计算好的数据传出来，
就完成 主业务逻辑层和子模块的业务分离，是一种解耦合效果。

7 耦合度

2015年11月14日 11:03

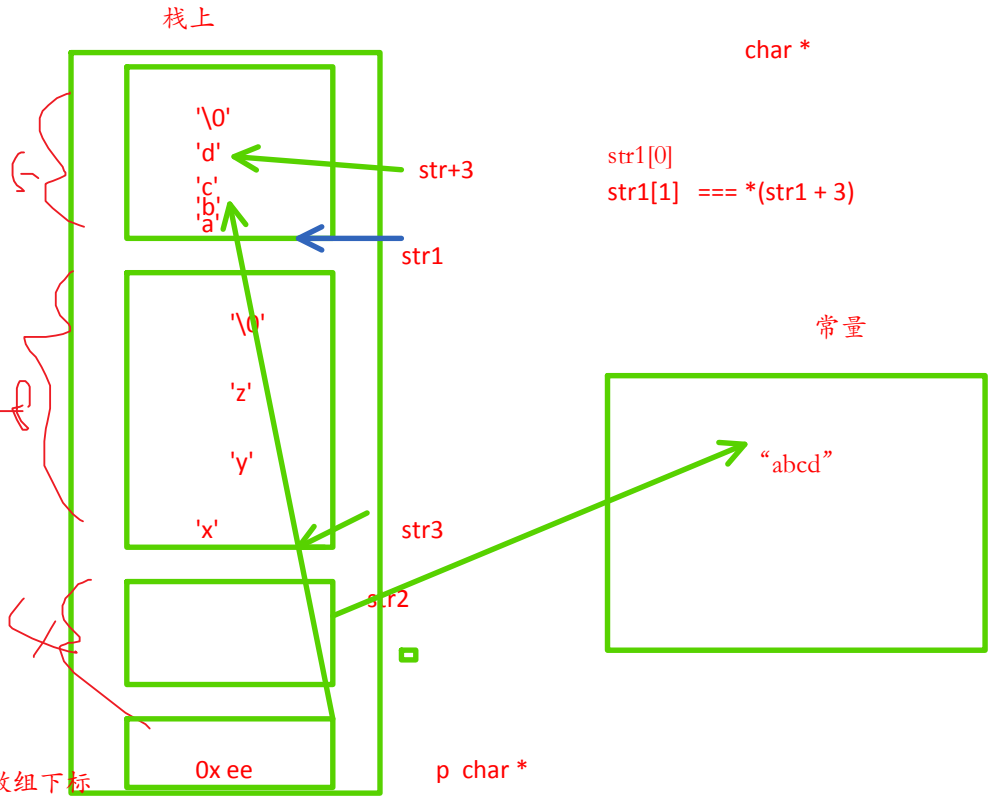
int funa(int, int, int , int ,it ,int)



8 数组名 和指针的区别

2015年11月14日 11:37

```
char str1[] = {'a', 'b', 'c', 'd','\0'}; 5
char str3[128] = { 'x', 'y', 'z' };
char *str2 = "abcd"; //4
```




```
char *p = NULL;
p = str1;
for (i = 0; i < len; i++, p++) {
      printf("%c\n", *p);
}

p++;

p = p +1;
```

```
for (i = 0; i < len; i++) {
      printf("%c\n", str1[i]); //通过数组下标
      [] 直接索引
}
```

通过不断改变指针的方向 来索引数组， 会比直接用数组名 索引速度要慢。
有不断的修改指针的方向 就是修改内存

```
p++;
str1++; //数组名 的方向 是不能够被改变的
      //数组名是 一个常量指针（方向改变不了）
      //栈是操作系统开辟和回收，是根据数组名来找到 一块连续内存的首地址， 如果首地址变量改变了。 将无法准确回收。
```

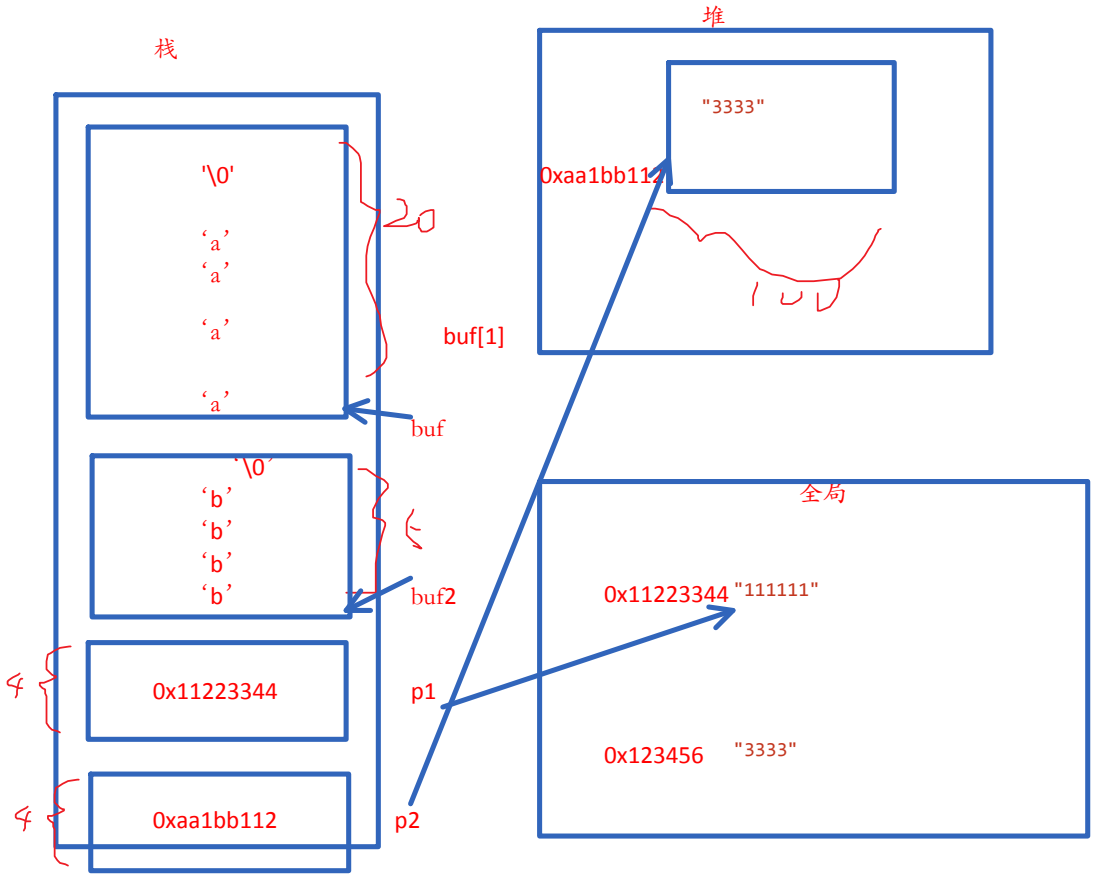
数组 和 指针

数组一般 在 栈上开辟内存
指针 一般在 堆上。

9 字符串的内存四区

2015年11月14日 14:46

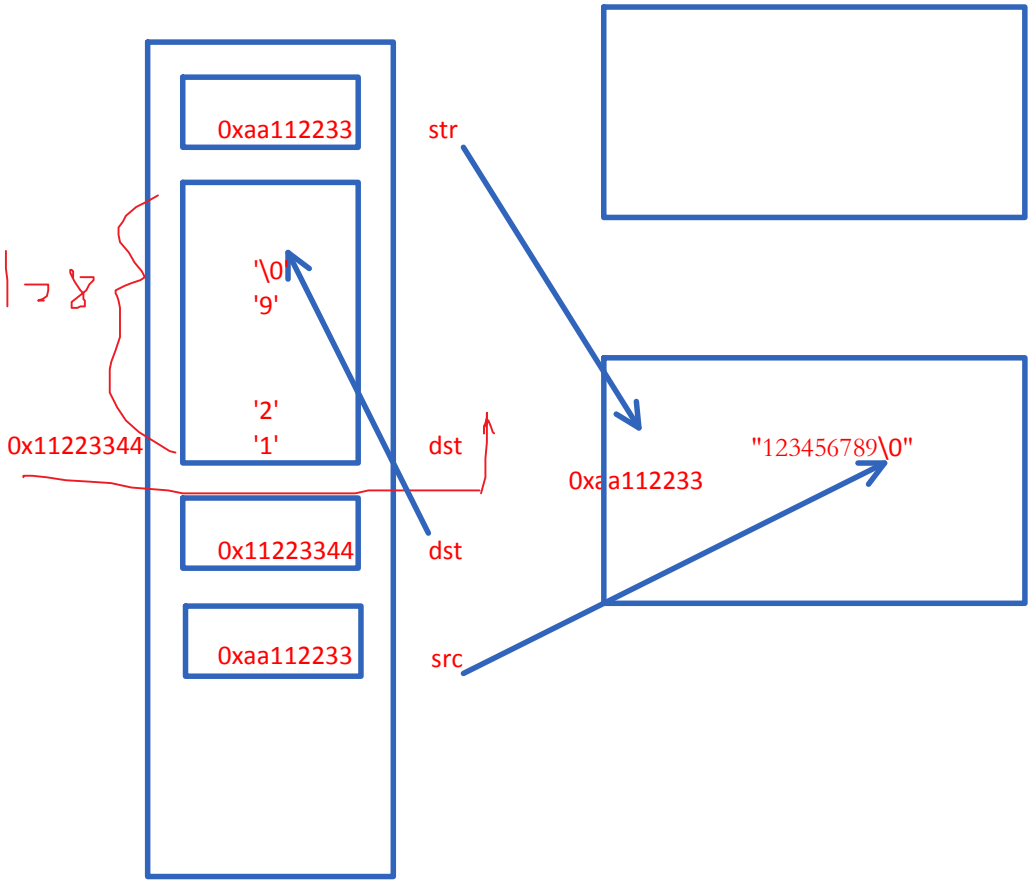
```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char buf[20]= "aaaa";
    char buf2[] = "bbbb";
    char *p1 = "111111";
    char *p2 = malloc(100);

    strcpy(p2, "3333");
    return 0;
}
```



10 str_copy

2015年11月14日 14:54

```
void str_copy(char *dst, char *src)
{
	for (; *src != '\0'; src++,
	dst++) {
		*dst = *src;
	}
}

int main(void)
{
	char *str = "123456789";
	char dst[128] = { 0 };
	str_copy(dst, str); //dst =
	dst,src = str
	return 0;
}
```



11 字符串中求子串

2015年11月14日 15:47

算出 "itcast" 的个数

"123123213itcastdsadjasjdosaitcastsd3121eitcast49342109321itcast"

12 两头堵模型

2015年11月14日 16:24

算出此字符串中 合法的字符串长度

key =		value

key = value

key =			value

key = value
key = value
key = value

q - p + 1 = 2

3

str[3] = '\0'

123

cnt = q - p + 1

13 字符串反转 递归思想

2015年11月14日 17:21