数组的存储，C语言数组的存储实质详解

在程序设计中，为了便于程序处理，通常把具有相同类型的若干变量按有序的形式组织在一起，这些按序排列的同类数据元素的集合称为数组。其中，集合中的每一个元素都相当于一个与数组同类型的变量；集合中的每一个元素用同一个名字和它在集合中的序号（下标）来区分引用。来看下面一个数组定义：

```
1. int a[5];
```

如图 1 所示，当定义一个数组a时，编译器根据指定的元素个数和元素的类型分配确定大小（元素类型大小×元素个数）的一块内存，并把这块内存的名字命名为 a，名字 a 一旦与这块内存匹配就不能再改变。其中，a[0]、a[1]、a[2]、a[3] 与 a[4] 都为 a 的元素，但并非元素的名字（数组的每一个元素都是没有名字的）。

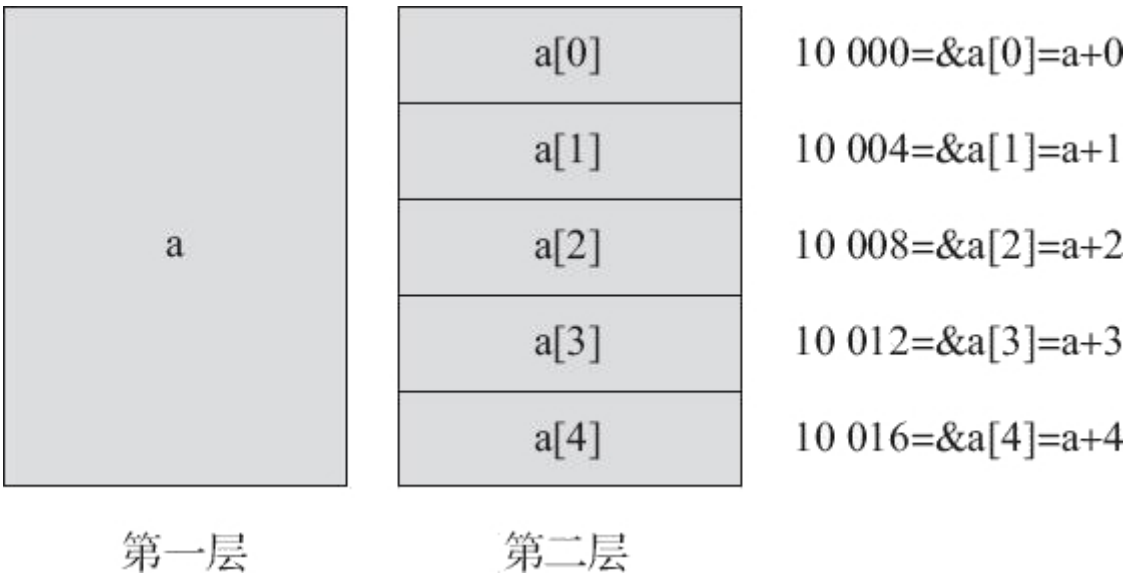

图 1 int[5]的存储结构

在 32 位系统中，由于 int 类型的数据占 4 字节单元，因此该数组 a 在内存中共占据连续的 4×5=20 字节单元，依次保存 a[0]、a[1]、a[2]、a[3] 与 a[4] 共 5 个元素。如果这里假设元素 a[0] 的地址是 10000，则元素 a[1] 的地址是 10000+1×4=10004; 元素 a[2] 的地址是 10000+2×4=10008; 元素 a[3] 的地址是 10000+3×4=10012; 元素 a[4] 的地址是 10000+4×4=10016。

由此可见，数组的存储具有如下特点：

- 索引从 0 开始。
- 数组在内存中占据连续的字节单元。
- 数组占据的字节单元数等于数组元素个数乘以该数组所属数据类型的数据占据的字节单元数（元素个数乘以元素类型大小）。
- 数组元素按顺序连续存放。

为了让大家更加清楚地看到数组的存储结构，继续看下面的示例代码：

```
1. int a[5];
2. printf("sizeof(a):%d\n",sizeof(a));
3. printf("sizeof(a[0]):%d\n",sizeof(a[0]));
4. printf("sizeof(a[5]):%d\n",sizeof(a[5]));
5. printf("sizeof(&a):%d\n",sizeof(&a));
6. printf("sizeof(&a[0]):%d\n",sizeof(&a[0]));
7. printf("---------------------------------\n");
8. printf("&a:%d\n",&a);
9. printf("&a[0]:%d\n",&a[0]);
10. printf("&a[1]:%d\n",&a[1]);
11. printf("&a[2]:%d\n",&a[2]);
12. printf("&a[3]:%d\n",&a[3]);
13. printf("&a[4]:%d\n",&a[4]);
```

对于上面的示例代码，在 32 位系统中：

- 对于sizeof(a)，sizeof(a)=sizeof(int)×5=4×5=20。
- 对于sizeof(a[0])，sizeof(a[0])=sizeof(int)=4。
- 对于sizeof(a[5])，sizeof(a[0])=sizeof(int)=4。

这里需要说明的是，因为 sizeof 是关键字，而不是函数（函数求值是在运行的时候，而关键字 sizeof 求值是在编译的时候），因此，虽然并不存在 a[5] 这个元素，但是这里也并没有真正访问 a[5]，而是仅仅根据数组元素的类型来确定其值。所以这里使用 a[5] 并不会出错，sizeof(a[5]) 的结果为 4。

对于 &a[0]，它表示取数组首元素 a[0] 的首地址；而对于 &a，表示取数组 a 的首地址。因此，&a[0] 的值与 &a 的值相同，sizeof(&a[0]) 与 sizeof(&a) 在 32 位系统下的结果都

为 4。

因此，运行上面的示例代码，运行结果为：

```
sizeof(a):20
sizeof(a[0]):4
sizeof(a[5]):4
sizeof(&a):4
sizeof(&a[0]):4
---------------------------------
&a:6356732
&a[0]:6356732
&a[1]:6356736
&a[2]:6356740
&a[3]:6356744
&a[4]:6356748
```

到现在为止，相信大家已经基本了解了一维数组的存储结构。或许这个时候你会问，那么二维数组及多维数组又是怎样存储的呢？其实，其原理与一维数组一样。下面，我们来定义一个 5 行 4 列的二维数组 a：

```
1. int a[5][4];
```

对于二维数组，它在逻辑上是由行和列组成的。因此，我们可以将上面的二维数组 a 分为三层来理解，如图 2 所示。

图 2

在图 2 中：
在第一层，将数组 a 看作一个变量，该变量的地址为 &a，长度为 sizeof(a)。因为数组的长度为元素数量乘以每个元素类型的大小，这里的二维数组 a 为 5 行 4 列共 20 个元素，每个元素占用 4 字节，所以变量 a 占用 80 字节。

在第二层，将数组 a 看作一个一维数组，由 a[0]、a[1]、a[2]、a[3] 与 a[4] 等 5 个元素组成。数组的首地址为 a 或 &a[0]（即数组首地址和第一个元素的地址相同，而每个数组元素的地址相差为 16，表示每个数组元素的长度为 16），使用 sizeof(a[0]) 可得到数组元素的长度。

在第三层，将第二层中的每个数组元素看作一个单独的数组。第二层中的每一个元素又由 4 个元素构成，如 a[0] 又由 a[0][0]、a[0][1]、a[0][2] 与 a[0][3] 等 4 个元素组成。

结合上面的分析来看下面的示例代码：

```
1. int main(void)
2. {
3.     int a[5][4];
4.     int i=0;
5.     int j=0;
6.     printf("sizeof(a):%d\n",sizeof(a));
7.     printf("sizeof(a[0]):%d\n",sizeof(a[0]));
8.     printf("sizeof(a[0][0]):%d\n",sizeof(a[0][0]));
9.     printf("---------------------------------\n");
10.      printf("sizeof(&a):%d\n",sizeof(&a));
11.      printf("sizeof(&a[0]):%d\n",sizeof(&a[0]));
12.      printf("sizeof(&a[0][0]):%d\n",sizeof(&a[0][0]));
13.      printf("---------------------------------\n");
14.      printf("&a:%d\n",&a);
15.      printf("&a[0]:%d\n",&a[0]);
16.      printf("&a[0][0]:%d\n",&a[0][0]);
17.      printf("---------------------------------\n");
18.      for(i=0;i<5;i++)
19.      {
```

```
20.         printf("&a[%d]:%d\n",i,&a[i]);
21.         for(j=0;j<4;j++)
22.         {
23.             printf("&a[%d][%d]:%d\n",i,j,&a[i][j]);
24.         }
25.     }
26.     return 0;
27. }
```

在上面的示例代码中，由于数组名代表的是数组首元素的首地址，因此下面的三行代码的输出结果都是相同的：

```
1. printf("&a:  %d\n",&a);
2. printf("&a[0]:%d\n",&a[0]);
3. printf("&a[0][0]:%d\n",&a[0][0]);
```

同时，当将 a[0] 作为一个数组名称时，该数组的首地址也就保存在 a[0] 中（这里 a[0] 作为一个整体看作数组名，而不是一个数组的元素）。因此，不用取地址运算符 &，直接输出 a[0] 的值也可得到数组的首地址，即下面的两行代码输出的结果是等价的：

```
1. printf("&a[0]:%d\n",&a[0]);
2. printf("&a[0]:%d\n",a[0]);
```

运行上面的示例代码，运行结果为：

```
sizeof(a):80
sizeof(a[0]):16
sizeof(a[0][0]):4
---------------------------------
sizeof(&a):4
sizeof(&a[0]):4
sizeof(&a[0][0]):4
---------------------------------
&a:6356664
&a[0]:6356664
&a[0][0]:6356664
---------------------------------
&a[0]:6356664
```

&a[0][0]:6356664
&a[0][1]:6356668
&a[0][2]:6356672
&a[0][3]:6356676
&a[1]:6356680
&a[1][0]:6356680
&a[1][1]:6356684
&a[1][2]:6356688
&a[1][3]:6356692
&a[2]:6356696
&a[2][0]:6356696
&a[2][1]:6356700
&a[2][2]:6356704
&a[2][3]:6356708
&a[3]:6356712
&a[3][0]:6356712
&a[3][1]:6356716
&a[3][2]:6356720
&a[3][3]:6356724
&a[4]:6356728
&a[4][0]:6356728
&a[4][1]:6356732
&a[4][2]:6356736
&a[4][3]:6356740

理解 &a[0] 和 &a 的区别

在对上面的数组示例分析的过程中，可以发现 &a[0] 和 &a 的值是相同的。但是要注意，尽管它们的结果相同，但其所表达的意义却完全不相同，这一点一定要注意。

因为数组名包含数组的首地址（即数组第一个元素的地址），或者说数组名指向数组的首地址（或第一个元素），所以，对于 &a，表示取数组 a 的首地址；而对于 &a[0]，它表示取数组首元素 a[0] 的首地址。这就好像陕西的省政府在西安，而西安的市政府同样也在西安。虽然两个政府机构都在西安，但其代表的意义完全不同。

理解数组名 a 作为右值和左值的区别

当数组名 a 作为右值的时候，其意义与 &a[0] 是一样的，代表的是数组首元素的首地址（注意，不是数组的首地址，这一点一定要区分开）。但是，这仅仅只是代表，编译器并没有为其分配一块内存空间来存放其地址，这一点就与指针有很大的差别。

同理，当数组名 a 作为左值的时候，代表的同样是数组的首元素的首地址。但是，这个地址开始的一块内存是一个总体（即数组一旦定义就会被分配一片连续的存储空间）。因此，我们只能访问数组的某个元素，而无法把数组当一个总体进行访问。也就是说我们可以将 a[0] 作为左值，而无法将 a 作为左值。