

作者：道哥，10+年嵌入式开发老兵，专注于：C/C++、嵌入式、Linux。

关注下方公众号，回复【书籍】，获取 Linux、嵌入式领域经典书籍；回复【PDF】，获取所有原创文章(PDF 格式)。

目录

别名是啥玩意？

方法1：反向注册

plugin.c 源文件

main.c 源文件

【关于作者】

方法2：嵌入汇编代码

plugin.c 源文件

main.c 源文件

小结

别人的经验，我们的阶梯！

别名是啥玩意？

在stackoverflow上看到一个有趣的话题：如何给一个变量设置一个别名？(How to assign to a variable an alias?)

The screenshot shows a Stack Overflow page for the question "How to assign to a variable an alias". The question was asked 8 years, 11 months ago and modified 4 years, 9 months ago, with 29k views. The user asks: "I want to call the same variable with a different name, how can I assign it an alias? Do I stick to using a macro, like #DEFINE Variable Alias". The answer provided is a summary of three scenarios: 1. I prefer to apply it in C, 2. I have a signal which can be more than one type of variable (temperature, distance, etc..), and 3. I want to assign more than one alias to that signal. The user mentions they are currently using functions in C for renaming and shows a code snippet: `int signal`.

stackoverflow About Products For Teams Search...

Home PUBLIC Questions Tags Users Companies COLLECTIVES Explore Collectives TEAMS

Stack Overflow for Teams – Start collaborating and sharing organizational knowledge.

Free

How to assign to a variable an alias

Asked 8 years, 11 months ago Modified 4 years, 9 months ago Viewed 29k times

I want to call the same variable with a different name, how can I assign it an alias?

Do I stick to using a macro, like

```
#DEFINE Variable Alias
```

In summary:

1. I prefer to apply it in C
2. I have a signal which can be more than one type of variable (temperature, distance, etc..)
3. I want to assign more than one alias to that signal

I'm currently doing it in C using functions as the method for renaming.

So given the variable: `int signal`

所谓的变量别名，就是通过通过不同的标识符，来表示同一个变量。

我们知道，变量名称是给程序员使用的。

在编译器的眼中，所有的变量都变成了地址。

请注意：这里所讨论的别名，仅仅是通过不同的标识符来引用同一个变量。

与强符号、弱符号没有关系，那是另一个话题。

在上面这个帖子中，作者首先想到的是通过宏定义，对变量进行重新命名。

这样的做法，将会在编译之前的预处理环节，把宏标识符替换为变量标识符。

在网友回复的答案中，大部分都是通过指针来实现：让不同的标识符指向同一个变量。

不管怎么说，这也算是一种别名了。

但是，这些答案有一个局限：这些代码必须一起进行编译才可以，否则就可能出现无法找到符号的错误信息。

现在非常流行插件编程，如果开发者想在插件中通过一个变量别名来引用主程序中的变量，这该如何处理呢？

本文提供两个方法来实现这个目的，并通过两个简单的示例代码来进行演示。

文末有示例代码的下载地址。

方法1：反向注册

之前我接触过一些CodeSys的代码，里面的代码质量真的是非常的高，特别是软件架构设计部分。

传说：CodySys 是工控界的 Android。

其中有个反向注册的想法，正好可以用在变量别名上面。

示例代码中一共有 2 个文件：main.c和plugin.c。

main.c中定义了一个全局变量数组，编译成可执行程序main。

plugin.c中通过一个别名来使用main.c中的全局变量。

plugin.c被编译成一个动态链接库，被可执行程序main动态加载(dlopen)。

在plugin.c中，提供一个函数func_init，当动态库被main dlopen之后，这个函数就被调用，并且把真正的全局变量的地址通过参数传入。

这样的话，在插件中就可以通过一个别名来使用真正的变量了（比如：修改变量的值）。

本质上，这仍然是通过指针来进行引用。

只不过利用动态注册的思想，把指针与变量的绑定关系在时间和空间上进行隔离。

plugin.c 源文件

```
#include <stdio.h>

int *alias_data = NULL;

void func_init(int *data)
{
    printf("libplugin.so: func_init is called. \n");
    alias_data = data;
}

void func_stage1(void)
{
    printf("libplugin.so: func_stage1 is called. \n");
    if (alias_data)
    {
        alias_data[0] = 100;
        alias_data[1] = 200;
    }
}
```

main.c 源文件

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

// defined in libplugin.so
typedef void (*pfunc_init)(int *);
typedef void (*pfunc_stage1)(void);

int data[100] = { 0 };

void main(void)
{
    data[0] = 10;
    data[1] = 20;

    printf("data[0] = %d \n", data[0]);
    printf("data[1] = %d \n", data[1]);

    // open libplugin.so
    void *handle = dlopen("./libplugin.so", RTLD_NOW);
    if (!handle)
    {
        printf("dlopen failed. \n");
        return;
    }
}
```

```
// get and call init function in libplugin.so
pfunc_init func_init = (pfunc_init) dlsym(handle, "func_init");
if (!func_init)
{
    printf("get func_init failed. \n");
    return;
}
func_init(data);

// get and call routine function in libplugin.so
pfunc_stage1 func_stage1 = (pfunc_stage1) dlsym(handle, "func_stage1");
if (!func_stage1)
{
    printf("get func_stage1 failed. \n");
    return;
}
func_stage1();

printf("data[0] = %d \n", data[0]);
printf("data[1] = %d \n", data[1]);

return;
}
```

编译指令如下：

```
gcc -m32 -fPIC --shared plugin.c -o libplugin.so
gcc -m32 -o main main.c -ldl
```

执行结果：

```
data[0] = 10
data[1] = 20
libplugin.so: func_init is called.
libplugin.so: func_stage1 is called.
data[0] = 100
data[1] = 200
```

可以看一下动态链接库的符号表：

```
readelf -s libplugin.so | grep data
```

```
7: 00002014 0 NOTYPE GLOBAL DEFAULT 23 _edata
14: 00002018 4 OBJECT GLOBAL DEFAULT 24 alias_data
48: 00002014 0 NOTYPE GLOBAL DEFAULT 23 _edata
52: 00002018 4 OBJECT GLOBAL DEFAULT 24 alias_data
```

可以看到alias_data标识符，并且是在本文件中定义的全局变量。

【关于作者】

号主：道哥，十多年的嵌入式开发老兵，专注于嵌入式 + Linux 领域，玩过单片机、搞过智能家居、研究过 PLC 和 工业机器人，项目开发经验非常丰富。

他的文章主要包括 C/C++、Linux操作系统、物联网、单片机和嵌入式这几个方面。

厚积薄发、换位思考，以读者的角度来总结文章。

每一篇输出，不仅仅是干货的呈现，更是引导你一步一步的深入思考，从底层逻辑来提升自己。

方法2：嵌入汇编代码

在动态加载的插件中使用变量别名，除了上面演示的动态注册的方式，还可以通过嵌入汇编代码来：设置一个全局标号来实现。

直接上示例代码：

plugin.c源文件

```
#include <stdio.h>

asm(".Global alias_data");
asm("alias_data = data");

extern int alias_data[];

void func_stage1(void)
{
    printf("libplugin.so: func_stage1 is called. \n");

    *(alias_data + 0) = 100;
    *(alias_data + 1) = 200;
}
```

main.c源文件

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

// defined in libplugin.so
typedef void (*pfunc_init)(int *);
typedef void (*pfunc_stage1)(void);
```

```

int data[100] = { 0 };

void main(void)
{
    data[0] = 10;
    data[1] = 20;

    printf("data[0] = %d \n", data[0]);
    printf("data[1] = %d \n", data[1]);

    // open libplugin.so
    void *handle = dlopen("./libplugin.so", RTLD_NOW);
    if (!handle)
    {
        printf("dlopen failed. \n");
        return;
    }

    // get and call routine function in libplugin.so
    pfunc_stage1 func_stage1 = (pfunc_stage1) dlsym(handle, "func_stage1");
    if (!func_stage1)
    {
        printf("get func_stage1 failed. \n");
        return;
    }
    func_stage1();

    printf("data[0] = %d \n", data[0]);
    printf("data[1] = %d \n", data[1]);

    return;
}

```

编译指令:

```

gcc -m32 -fPIC --shared plugin.c -o libplugin.so
gcc -m32 -rdynamic -o main main.c -ldl

```

执行结果:

```

data[0] = 10
data[1] = 20
libplugin.so: func_stage1 is called.
data[0] = 100
data[1] = 200

```

也来看一下libplugin.so中的符号信息:

```
readelf -s libplugin.so | grep data
```

```
5: 00000000 0 NOTYPE GLOBAL DEFAULT UND data
8: 00002014 0 NOTYPE GLOBAL DEFAULT 23 _edata
48: 00002014 0 NOTYPE GLOBAL DEFAULT 23 _edata
54: 00000000 0 NOTYPE GLOBAL DEFAULT UND data
```

小结

这篇文档通过两个示例代码，讨论了如何在插件中(动态链接库)，通过别名来访问真正的变量。

不知道您会不会有这样的疑问：直接使用extern来声明一下外部定义的变量不就可以了，何必这么麻烦？

道理是没错！

但是，在一些比较特殊的领域或场景中(比如一些二次开发中)，这样的需求是确实存在的，而且是强需求。

如果你有任何疑问，或者文中有任务错误，欢迎留言讨论、指正。

----- End -----

在公众号【IOT物联网小镇】后台回复关键字：20522，即可获取示例代码的下载地址。

既然看到这里了，如果觉得不错，请您随手点个【赞】和【在看】吧！

如果转载本文，文末务必注明：“转自微信公众号：IOT物联网小镇”。

推荐阅读

- 【1】《Linux 从头学》系列文章
- 【2】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
- 【3】原来gdb的底层调试原理这么简单
- 【4】Linux中对【库函数】的调用进行跟踪的3种【插桩】技巧
- 【5】内联汇编很可怕吗？看完这篇文章，终结它！
- 【6】gcc编译时，链接器安排的【虚拟地址】是如何计算出来的？
- 【7】GCC 链接过程中的【重定位】过程分析
- 【8】Linux 动态链接过程中的【重定位】底层原理

其他系列专辑：精选文章、应用程序设计、物联网、C语言。

