[DEVICE\_ATTR的使用](http://blog.csdn.net/njuitjf/article/details/16849333)

使用DEVICE\_ATTR，可以在sys fs中添加“文件”，通过修改该文件内容，可以实现在运行过程中动态控制device的目的。  
类似的还有DRIVER\_ATTR，BUS\_ATTR，CLASS\_ATTR。  
这几个东东的区别就是，DEVICE\_ATTR对应的文件在/sys/devices/目录中对应的device下面。  
而其他几个分别在driver，bus，class中对应的目录下。  
这次主要介绍DEVICE\_ATTR，其他几个类似。  
在documentation/driver-model/Device.txt中有对DEVICE\_ATTR的详细介绍，这儿主要说明使用方法。

先看看DEVICE\_ATTR的原型：  
DEVICE\_ATTR(\_name, \_mode, \_show, \_store)  
\_name：名称，也就是将在sys fs中生成的文件名称。  
\_mode：上述文件的访问权限，与普通文件相同，UGO的格式。  
\_show：显示函数，cat该文件时，此函数被调用。  
\_store：写函数，echo内容到该文件时，此函数被调用。

看看我们怎么填充这些要素：  
名称可以随便起一个，便于记忆，并能体现其功能即可。  
模式可以为只读0444，只写0222，或者读写都行的0666。当然也可以对User\Group\Other进行区别。  
显示和写入函数就需要实现了。

显示函数的一般实现：  
static ssize\_t xxx\_show(struct device \*dev,  
 struct device\_attribute \*attr, char \*buf)  
{  
 return scnprintf(buf, PAGE\_SIZE, "%d\n", dma\_dump\_flag);  
}  
实现相对简单，调用了个很阳春的scnprintf，把数据放到buf中，就算大功告成了。  
至于buf中的内容怎么显示出来，这个先略过。

写入函数的一般实现：  
static ssize\_t xxx\_store(struct device \*dev,  
 struct device\_attribute \*attr, const char \*buf, size\_t count)  
{  
 unsigned long num;  
 if (strict\_strtoul(buf, 0, &num))  
  return -EINVAL;  
 if (num < 0)  
  return -EINVAL;  
 mutex\_lock(&xxx\_lock);  
 dma\_dump\_flag = num;  
 mutex\_unlock(&xxx\_lock);  
 return count;  
}  
也挺直白，就不细说了。  
其中加了个lock进行互斥。

函数名中的后缀\_show和\_store当然不是必须的。  
只是便于标识。

DEVICE\_ATTR的定义例子：  
static DEVICE\_ATTR(xxx, 0666, xxx\_show, xxx\_store);  
该代码可以防止文件的任何位置，只要别引起编译错误！

是不是这样就搞定了？  
当然没有，还需要调用函数device\_create\_file来传教sys fs中的文件。  
调用方法：  
device\_create\_file(&pdev->dev, &dev\_attr\_xxx);  
文件不是创建在某个device目录下么，pdev->dev就是该device。  
dev\_attr\_xxx就是在xxx前加上dev\_attr\_，好像是废话，不过现实就是这样。  
开始还找了半天，dev\_attr\_xxx在哪儿定义？  
最终发现这儿就是它唯一出现的地方。

device\_create\_file的调用例子：  
  ret = device\_create\_file(&pdev->dev, &dev\_attr\_xxx);  
  if (ret != 0) {  
   dev\_err(&pdev->dev,  
    "Failed to create xxx sysfs files: %d\n", ret);  
   return ret;  
  }

这个代码最好放在device的probe函数中。  
原因么，在documentation/driver-model/Device.txt中有说明。

下面看看DEVICE\_ATTR的定义：  
#define DEVICE\_ATTR(\_name, \_mode, \_show, \_store) \  
struct device\_attribute dev\_attr\_##\_name = \_\_ATTR(\_name, \_mode, \_show, \_store)  
dev\_attr\_##\_name！！！！！  
终于找到dev\_attr\_xxx定义的地方了！

#define \_\_ATTR(\_name,\_mode,\_show,\_store) { \  
 .attr = {.name = \_\_stringify(\_name), .mode = \_mode }, \  
 .show = \_show,     \  
 .store = \_store,     \  
}

device\_attribute定义：  
struct device\_attribute {  
 struct attribute attr;  
 ssize\_t (\*show)(struct device \*dev, struct device\_attribute \*attr,  
   char \*buf);  
 ssize\_t (\*store)(struct device \*dev, struct device\_attribute \*attr,  
    const char \*buf, size\_t count);  
};

DEVICE\_ATTR的功能就是定义一个device\_attribute结构体对象。  
device\_create\_file利用该对象在device下创建文件。

/\*\*  
 \* device\_create\_file - create sysfs attribute file for device.  
 \* @dev: device.  
 \* @attr: device attribute descriptor.  
 \*/  
int device\_create\_file(struct device \*dev,  
         const struct device\_attribute \*attr)  
{  
 int error = 0;  
 if (dev)  
  error = sysfs\_create\_file(&dev->kobj, &attr->attr);  
 return error;  
}

/\*\*  
 \* sysfs\_create\_file - create an attribute file for an object.  
 \* @kobj: object we're creating for.   
 \* @attr: attribute descriptor.  
 \*/

int sysfs\_create\_file(struct kobject \* kobj, const struct attribute \* attr)  
{  
 BUG\_ON(!kobj || !kobj->sd || !attr);

 return sysfs\_add\_file(kobj->sd, attr, SYSFS\_KOBJ\_ATTR);  
}

sd的类型为struct sysfs\_dirent。  
/\*  
 \* sysfs\_dirent - the building block of sysfs hierarchy.  Each and  
 \* every sysfs node is represented by single sysfs\_dirent.  
 \*  
 \* As long as s\_count reference is held, the sysfs\_dirent itself is  
 \* accessible.  Dereferencing s\_elem or any other outer entity  
 \* requires s\_active reference.  
 \*/  
struct sysfs\_dirent {  
 atomic\_t  s\_count;  
 atomic\_t  s\_active;  
#ifdef CONFIG\_DEBUG\_LOCK\_ALLOC  
 struct lockdep\_map dep\_map;  
#endif  
 struct sysfs\_dirent \*s\_parent;  
 struct sysfs\_dirent \*s\_sibling;  
 const char  \*s\_name;

 const void  \*s\_ns; /\* namespace tag \*/  
 union {  
  struct sysfs\_elem\_dir  s\_dir;  
  struct sysfs\_elem\_symlink s\_symlink;  
  struct sysfs\_elem\_attr  s\_attr;  
  struct sysfs\_elem\_bin\_attr s\_bin\_attr;  
 };

 unsigned int  s\_flags;  
 unsigned short  s\_mode;  
 ino\_t   s\_ino;  
 struct sysfs\_inode\_attrs \*s\_iattr;  
};

int sysfs\_add\_file(struct sysfs\_dirent \*dir\_sd, const struct attribute \*attr,  
     int type)  
{  
 return sysfs\_add\_file\_mode(dir\_sd, attr, type, attr->mode);  
}

int sysfs\_add\_file\_mode(struct sysfs\_dirent \*dir\_sd,  
   const struct attribute \*attr, int type, mode\_t amode)  
{  
 umode\_t mode = (amode & S\_IALLUGO) | S\_IFREG;  
 struct sysfs\_addrm\_cxt acxt;  
 struct sysfs\_dirent \*sd;  
 int rc;

 // 分配空间，并初始化部分成员。  
 sd = sysfs\_new\_dirent(attr->name, mode, type);  
 if (!sd)  
  return -ENOMEM;  
 sd->s\_attr.attr = (void \*)attr;  
/\*  
 \* Initialize a lock instance's lock-class mapping info:  
 \*/  
 sysfs\_dirent\_init\_lockdep(sd);

/\*\*  
 \* sysfs\_addrm\_start - prepare for sysfs\_dirent add/remove  
 \* @acxt: pointer to sysfs\_addrm\_cxt to be used  
 \* @parent\_sd: parent sysfs\_dirent  
 \*  
 \* This function is called when the caller is about to add or  
 \* remove sysfs\_dirent under @parent\_sd.  This function acquires  
 \* sysfs\_mutex.  @acxt is used to keep and pass context to  
 \* other addrm functions.  
 \*  
 \* LOCKING:  
 \* Kernel thread context (may sleep).  sysfs\_mutex is locked on  
 \* return.  
 \*/  
 sysfs\_addrm\_start(&acxt, dir\_sd);  
   
/\*\*  
 \* sysfs\_add\_one - add sysfs\_dirent to parent  
 \* @acxt: addrm context to use  
 \* @sd: sysfs\_dirent to be added  
 \*  
 \* Get @acxt->parent\_sd and set sd->s\_parent to it and increment  
 \* nlink of parent inode if @sd is a directory and link into the  
 \* children list of the parent.  
 \*  
 \* This function should be called between calls to  
 \* sysfs\_addrm\_start() and sysfs\_addrm\_finish() and should be  
 \* passed the same @acxt as passed to sysfs\_addrm\_start().  
 \*  
 \* LOCKING:  
 \* Determined by sysfs\_addrm\_start().  
 \*  
 \* RETURNS:  
 \* 0 on success, -EEXIST if entry with the given name already  
 \* exists.  
 \*/  
 rc = sysfs\_add\_one(&acxt, sd);  
   
/\*\*  
 \* sysfs\_addrm\_finish - finish up sysfs\_dirent add/remove  
 \* @acxt: addrm context to finish up  
 \*  
 \* Finish up sysfs\_dirent add/remove.  Resources acquired by  
 \* sysfs\_addrm\_start() are released and removed sysfs\_dirents are  
 \* cleaned up.  
 \*  
 \* LOCKING:  
 \* sysfs\_mutex is released.  
 \*/  
 sysfs\_addrm\_finish(&acxt);

 if (rc)  
  sysfs\_put(sd);

 return rc;  
}

基本上知道是怎么回事了。  
暂时先到这，就不再深入了。