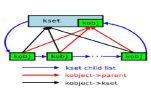
努力成为 linux kernel hacker 的人李万鹏原创作品，转载请标明出处
http://blog.csdn.net/woshixingaaa/archive/2011/05/05/6396618.aspx
如果哪里有理解不对的请指教，文章引用的内核源码版本为 2.6.29.1 的。
建立设备模型主要为了管理方便。最初引入设备模型是为了电源管理。建立一个全局的设备树（device tree），当系统进入休眠时，系统可以通过这棵树找到所有的设备，随时让他们挂起（suspend）或者唤醒（resume）。
2.6 版内核提供的功能：
电源管理和系统关机
　　完成这些工作需要对一些系统结构的理解。比如一个 USB 宿主适配器，在处理完所有与其连接的设备前是不能被关闭的。设备模型使得操作系统能够以正确的顺序遍历硬件。
与用户空间通信
　　sysfs 虚拟文件系统的实现与设备模型密切相关，并且向外界展示了它所表示的结构。向用户空间所提供的系统信息，以及改变操作参数的接口，将越来越多的通过 sysfs 实现，也就是说通过设备模型实现。
热插拔设备
　　内核中的热插拔机制可以处理热插拔设备，特别是能够与用户空间进行关于热插拔设备的通信，而这种机制也是通过热插拔管理的。
设备类型
　　把设备分门别类有助于设备的管理与使用。比如要找 USB 鼠标，只要去 classes/input/里去找就可以了，而不必关心这个鼠标是接到哪个 USB 主机控制器的哪个 Hub 的第几个端口上。
对象生命周期
　　得有一个好的机制来实现设备生命周期的管理。比如把 USB 鼠标拔了之后，全局设备树和 sysfs 里面得相应去掉。
设备底层模型：
　　Linux 设备模型的底层是数据结构 kobject，内核用 kobject 结构将各个对象连接起来组成一个分层的结构体系，从而与模块化的子系统相匹配。一个 kset 是嵌入相同类型结构的 kobject 集合。kset 和他的 kobject 的关系与下图类似，请记住：

- ü　　在图中所有被包含的 kobject，实际上是被嵌入到其他类型中的，甚至可能是其他的 kset。
- ü　　一个 kobject 的父节点不一定是包含它的 kset。



Kobject 是组成设备模型的基本结构，最初他只是被理解为一个简单的引用计数，但是随着时间的推移，他的任务越来越多，因此也有了许多成员，他的结构体如下：

```
1.  struct kobject {
2.      const char      *name;
3.      struct list_head    entry;      //挂接到所在 kset 中去的单元
4.      struct kobject      *parent; //指向父对象的指针
5.      struct kset     *kset;      //所属 kset 的指针
6.      struct kobj_type    *ktype;     //指向其对象类型描述符的指针
7.      struct sysfs_dirent *sd;    //sysfs 文件系统中与该对象对应的目录实体的指针
8.      struct kref     kref;
9.      unsigned int state_initialized:1;
10.     unsigned int state_in_sysfs:1;
11.     unsigned int state_add_uevent_sent:1;
12.     unsigned int state_remove_uevent_sent:1;
13.     unsigned int uevent_suppress:1;
14.};
```

name 指向设备的名字，entry，parent，kset 就是用来形成树状结构的指针。Kobj_type *type 用来表示该 kobject 的类型，struct sysfs_dirent 类型的指针指向了该 kobject 在 sysfs 中的目录实体，sysfs 中每一个 dentry 都会对应一个 sysfs_dirent 结构。每个在内核中注册的 kobject 对象都对应于 sysfs 文件系统中的一个目录。/sys 是专为 Linux 设备模型建立的，kobject 是帮助建立/sys 文件系统的。每当我们新增一个 kobject 结构时，同时会在/sys 下创建一个目录。这里隐藏了如下程序调用流程：kobject_add()->kobject_add_varg()->kobject_add_internal()->create_dir()->sysfs_new_dirent()。在 sysfs_new_dirent()函数中通过 slab 分配了一个 dirent（至于什么是 dirent 会在<Linux 设备模型（下）之 sysfs 文件系统>中讲解），并返回给一个指向这个 dirent 的指针 sd 给 create_dir(),在 create_dir()函数中有这么一句：sd->s_dir.kobj = kobj;也就是让 dirent 的一个成员的域指向了他所对应的 kobject，kobject 中 struct sysfs_dirent *sd;又指向了 dirent，所以 kobject 与 sysfs 死死的拥抱在一起，为了幸福的明天。在 kobject_del()函数中会调用 sysfs_remove_dir()，sysfs_remove_dir()中有这么一句：kobj->sd = NULL;表示 kobject 与 sysfs_dirent 的婚姻破裂了。

kobject 的接口函数：

```
1.  void kobject_init(struct kobject *kobj);
```

kobject 初始化函数，设置 kobject 引用计数为 1。

```
1.  int kobject_set_name(struct kobject *kobj, const char *format, ...);
```

设置 kobject 的名字。

```
1. struct kobject *kobject_get(struct kobject *kobj);
2. struct kobject *kobject_put(struct kobject *kobj);
```

减少和增加 kobject 的引用计数。

```
1. extern int __must_check kobject_init_and_add(struct kobject *kobj,
2.                 struct kobj_type *ktype,
3.                 struct kobject *parent,
4.                 const char *fmt, ...);
```

kobject 注册函数，该函数只是 kobjec_init 和 kobject_add_varg 的简单组合。旧内核称为

```
1. extern int kobject_register(struct kobject *kobj);
```

```
1. void kobject_del(struct kobject *kobj);
```

从 Linux 设备层次中（hierarchy）中删除 kobj 对象。

```
1. struct kset {
2.     struct list_head list;          //用于连接该 kset 中所有 kobject 的链表头
3.     spinlock_t list_lock;           //用于互斥访问
4.     struct kobject kobj;            //嵌入的 kobject
5.     struct kset_uevent_ops *uevent_ops;
6. };
```

包含在 kset 中的所有 kobject 被组织成一个双向循环链表，list 真是该链表的链表头。kset 数据结构还内嵌了一个 kobject 对象（由 kobj 表示），所有属于这个 kset 的 kobject 对象的 parent 域均指向这个内嵌的对象。此外，kset 还依赖于 kobj 维护引用计数：kset 的引用计数实际上就是内嵌的 kobject 对象的引用计数。

```
1. struct kobj_type {
2.     void (*release)(struct kobject *kobj);
3.     const struct sysfs_ops *sysfs_ops;
4.     struct attribute **default_attrs;
5.     const struct kobj_ns_type_operations *(*child_ns_type)(struct kobject *kobj);
6.     const void *(*namespace)(struct kobject *kobj);
7. };
```

释放 kobject 使用 release 函数，release 函数并没有包含在 kobject 自身内，他包含在与 kobject 相关联的 kobj_type 中。sysfs_ops 是指向如何读写的函数的指针。

```
1. struct sysfs_ops {
```

```
    ssize_t (*show)(struct kobject *, struct attribute *,char *);
    ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
};
```

show 相当于 read,store 相当于 write。
struct attribute **default_attrs;是属性数组。在 sysfs 中，kobject 对应目录，kobject 的属性对应这个目录下的文件。调用 show 和 store 函数来读写文件，就可以得到属性中的内容。

一个热插拔事件是从内核空间发送到用户空间的通知。它表明系统配置出现了变化。无论 kobject 被创建还是被删除，都会产生这种事件。比如，当数码相机通过 USB 线缆插入到系统时。热插拔事件会导致对/sbin/hotplug 程序的调用，该程序通过加载驱动程序，创建设备节点，挂装分区，或者其他正确的动作来响应。对热插拔事件的控制由保存在 结构体中的函数完成：

```
struct kset_uevent_ops {
    int (*filter)(struct kset *kset, struct kobject *kobj);
    const char *(*name)(struct kset *kset, struct kobject *kobj);
    int (*uevent)(struct kset *kset, struct kobject *kobj,
            struct kobj_uevent_env *env);
};
```

我们可以在 kset 结构的 hotplug_ops 成员中发现指向这个结构的指针。如果在 kset 中不包含一个指定的 kobject，内核将在分层结构中进行搜索（通过 parent 指针），直到找到一个包含有 kset 的 kobject 为止，然后使用这个 kset 的热插拔操作。下面是一个测试的程序：
kobject.c

```
#include <linux/device.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/string.h>
#include <linux/sysfs.h>
#include <linux/stat.h>

MODULE_AUTHOR("David Xie");
MODULE_LICENSE("Dual BSD/GPL");

void obj_test_release(struct kobject *kobject);
ssize_t kobj_test_show(struct kobject *kobject, struct attribute *attr,char *buf);
ssize_t kobj_test_store(struct kobject *kobject,struct attribute *attr,const char *buf, size_t count
    );

struct attribute test_attr = {
```

```
17.         .name = "kobj_config",
18.         .mode = S_IRWXUGO,
19.};
20.
21.static struct attribute *def_attrs[] = {
22.         &test_attr,
23.         NULL,
24.};
25.
26.
27.struct sysfs_ops obj_test_sysops =
28.{
29.         .show = kobj_test_show,
30.         .store = kobj_test_store,
31.};
32.
33.struct kobj_type ktype =
34.{
35.         .release = obj_test_release,
36.         .sysfs_ops=&obj_test_sysops,
37.         .default_attrs=def_attrs,
38.};
39.
40.void obj_test_release(struct kobject *kobject)
41.{
42.         printk("eric_test: release ./n");
43.}
44.
45.ssize_t kobj_test_show(struct kobject *kobject, struct attribute *attr,char *buf)
46.{
47.         printk("have show./n");
48.         printk("attrname:%s./n", attr->name);
49.         return strlen(attr->name)+2;
50.}
51.
52.ssize_t kobj_test_store(struct kobject *kobject,struct attribute *attr,const char *buf, size_t count
    )
53.{
54.         printk("havestore/n");
55.         printk("write: %s/n",buf);
56.         return count;
57.}
58.
59.struct kobject kobj;
60.static int kobj_test_init()
61.{
62.         printk("kboject test init./n");
63.         kobject_init_and_add(&kobj,&ktype,NULL,"kobject_test");
```

```
64.     return 0;
65.}
66.
67.static int kobj_test_exit()
68.{
69.     printk("kobject test exit./n");
70.     kobject_del(&kobj);
71.     return 0;
72.}
73.
74.module_init(kobj_test_init);
75.module_exit(kobj_test_exit);
```

kset.c

```
1.  #include <linux/device.h>
2.  #include <linux/module.h>
3.  #include <linux/kernel.h>
4.  #include <linux/init.h>
5.  #include <linux/string.h>
6.  #include <linux/sysfs.h>
7.  #include <linux/stat.h>
8.  #include <linux/kobject.h>
9.
10.MODULE_AUTHOR("David Xie");
11.MODULE_LICENSE("Dual BSD/GPL");
12.
13.struct kset kset_p;
14.struct kset kset_c;
15.
16.int kset_filter(struct kset *kset, struct kobject *kobj)
17.{
18.     printk("Filter: kobj %s./n",kobj->name);
19.     return 1;
20.}
21.
22.const char *kset_name(struct kset *kset, struct kobject *kobj)
23.{
24.     static char buf[20];
25.     printk("Name: kobj %s./n",kobj->name);
26.     sprintf(buf,"%s","kset_name");
27.     return buf;
28.}
29.
30.int kset_uevent(struct kset *kset, struct kobject *kobj,struct kobj_uevent_env *env)
31.{
32.     int i = 0;
33.     printk("uevent: kobj %s./n",kobj->name);
34.
```

```
35.     while( i < env->envp_idx){
36.         printk("%s./n",env->envp[i]);
37.         i++;
38.     }
39.
40.     return 0;
41.}
42.
43.struct kset_uevent_ops uevent_ops =
44.{
45.     .filter = kset_filter,
46.     .name   = kset_name,
47.     .uevent = kset_uevent,
48.};
49.
50.int kset_test_init()
51.{
52.     printk("kset test init./n");
53.     kobject_set_name(&kset_p.kobj,"kset_p");
54.     kset_p.uevent_ops = &uevent_ops;
55.     kset_register(&kset_p);
56.
57.     kobject_set_name(&kset_c.kobj,"kset_c");
58.     kset_c.kobj.kset = &kset_p;
59.     kset_register(&kset_c);
60.     return 0;
61.}
62.
63.int kset_test_exit()
64.{
65.     printk("kset test exit./n");
66.     kset_unregister(&kset_p);
67.     kset_unregister(&kset_c);
68.     return 0;
69.}
70.
71.module_init(kset_test_init);
72.module_exit(kset_test_exit);
```

测试效果：

```
1. root@hacker:/home/hacker/kobject# insmod kset.ko
2. root@hacker:/home/hacker/kobject# dmesg
3. [  866.344079] kset test init.
4. [  866.344090] Filter: kobj kset_c.
5. [  866.344092] Name: kobj kset_c.
6. [  866.344097] uevent: kobj kset_c.
7. [  866.344099] ACTION=add.
```

```
[  866.344101] DEVPATH=/kset_p/kset_c.
[  866.344103] SUBSYSTEM=kset_name.

root@hacker:/home/hacker/kobject# rmmod kset
root@hacker:/home/hacker/kobject# dmesg
[  892.202071] kset test exit.
[  892.202075] Filter: kobj kset_c.
[  892.202077] Name: kobj kset_c.
[  892.202083] uevent: kobj kset_c.
[  892.202085] ACTION=remove.
[  892.202087] DEVPATH=/kset_p/kset_c.
[  892.202089] SUBSYSTEM=kset_name.
```

如果将 kset_c.kobj.kset = &kset_p;这行注释掉，也就是不产生热插拔事件，效果如下：

```
root@hacker:/home/hacker/kobject# insmod kset.ko
root@hacker:/home/hacker/kobject# dmesg
[   94.146759] kset test init.
```

无论什么时候，当内核要为指定的 kobject 产生事件时，都要调用 filter 函数。如果 filter 返回 0，将不产生事件，这里将返回值改为 0，看效果：

```
root@hacker:/home/hacker/kobject# insmod kset.ko
root@hacker:/home/hacker/kobject# dmesg
[  944.457502] kset test init.
[  944.457535] Filter: kobj kset_c.

root@hacker:/home/hacker/kobject# rmmod kset
root@hacker:/home/hacker/kobject# dmesg
[  962.514146] kset test exit.
```

下边是 kobject 的测试效果：

```
root@hacker:/home/hacker/kobject# insmod kobject.ko
root@hacker:/home/hacker/kobject# dmesg
[ 1022.694855] kboject test init.

root@hacker:/home/hacker/kobject# rmmod kobject
root@hacker:/home/hacker/kobject# dmesg
[ 1056.650200] kobject test exit.

root@hacker:/sys/kobject_test# ls
kobj_config
root@hacker:/sys/kobject_test# cat kobj_config
root@hacker:/sys/kobject_test# dmesg
[ 1280.545220] have show.
[ 1280.545226] attrname:kobj_config.
root@hacker:/sys/kobject_test# echo "aha" > kobj_config
root@hacker:/sys/kobject_test# dmesg
[ 1280.545220] have show.
[ 1280.545226] attrname:kobj_config.
```

```
19.[ 1295.622228] havestore
20.[ 1295.622235] write: aha
```

分享到：