kobject

Documentation/kobject.txt

Everything you never wanted to know about kobjects, ksets, and ktypes

Greg Kroah-Hartman <gregkh@linuxfoundation.org>

Based on an original article by Jon Corbet for lwn.net written October 1,
2003 and located at http://lwn.net/Articles/51437/

Last updated December 19, 2007

Translator: Bill Wang(Liang) (bill.liangwlw@gmail.com,https://github.com/billwangwl)

难以理解driver model，以及设备模型所基于的kobject，的部分原因是，没有明确的可以开始学习的地方。要使用kobject，需要了解额外的几个类型，这几个类型又相互引用。这个文章为了帮助更好的学习，我们会从不同的角度来考虑问题，首先看几个粗略的定义，然后再添加足够的细节。为了达到这个目的，我们从定义开始：

--kobject是struct kobject的object。kobjects有一个名字和一个引用计数。一个kobject都有一个父kobject(这样kobject可以有层次），一个特定的type，通常，也会有在sysfs中对应的目录。

单单kobject没有什么意义，更关注的是包含kobject的类型。

任何结构不应该包含超过一个的kobject。如果超过了一个，这个结构的引用计数就会乱掉了。

--**ktype是包含kobject的结构的类型**。每个包含kobject的结构都该有一个对应的ktype。这个ktype决定了当kobject构建和销毁时会发生什么。

--kset是一组kobject。同一个kset的kobject可能是同一个ktype，也可以不同的ktype。**kset是集合kobject的基础的容器类型**。kset有自己的kojects，但是你可以完全不用关注kset自身的kobject，kset会自己处理这些kobject。

如果sysfs的一个目录都是子目录，通常这些目录都对应了同一个kset的kobject。

接下来我们从下到上的方式来看看如何处理这些类型。

Embedding kobjects

很少有kernel代码会生成一个单独的kobject，除了后续要说明的一个特例。相反，kobject会被嵌入到其他结构中，用于控制其他复杂的对象。如果你习惯于面向对象，kobject可以被认为是一个顶层基类，其他的类都从kobject派生。kobject实现了公共的部分。C语言没有直接实现继承的语法，所以用结构嵌入的实现继承关系。

（题外话，list_head也是类似的实现）

以dirvers/uio/uio.c中uio的代码为例：

```
struct uio_map {
        struct kobject kobj;
        struct uio_mem *mem;
};
```

这个结构体展示了一种是用kobj的方法。这种方法带来另外一个问题就是，如果找到嵌入kobject的对象？不要用诡异的方法，是用<linux/kernel.h>中定义的container_of()方法。

```
container_of(pointer, type, member)
```

查看http://www.kroah.com/log/linux/container_of.html 来了解container_of的方法。

Initialization of kobjects

使用kobject就要进行初始化。kobject_init()会完成kobject的初始化工作：

```
void kobject_init(struct kobject *kobj, struct kobj_type *ktype);
```

每一个kobject都要有一个ktype。在调用kobj_init()之后，需要将kobject注册到sysfs中，这个通过kobj_add来完成。

```
int kobject_add(struct kobject *kobj, struct kobjct *parent, const char *fmt, ...);
```

这个函数会通过parent建立起kobject的层次关系，通过格式化的字符串来设置名字。***如果kobject属于某个kset，那么kobj->kset必须在kobject_add()之前调用。如果如果某个koject是kset自身的kobject，那么在kobject_add()中的parent参数要设置为NULL，这样这个kobject的parent就是kset自身。***

kobject的名字在添加到sysfs时指定，不能直接访问名字。如果需要修改，则需要通过

```
int kobject_name(struct kobject *kobj, const char *new_name);
```

kobject_name不会进行检查，所以调用者需要进行恰当的检查。

有一个kobject_set_name()的方法，但是该方法已经过时不再使用。要访问kobject名字，需要通过

```
const char *kobject_name(const struct kobject *kobj);
```

有一个简便的方法可以完成初始化和向系统添加：

```
int kobject_init_and_add(struct kobject *kobj, struct kobj_type *ktype, struct kobject *parent, const char *fmt, ...)
```

Uevents

当kobject注册kobject core之后，要通知系统kobject已经注册，通过：

```
int kobject_uevent(struct kobject *kobj, enum kobject_action action);
```

在第一次添加到kernel中后，该函数用KOBJ_ADD来通知。当kobject有属性或者有子kobject初始化后，就应该调用该函数，这样，用户空间就得到通知。

当kobject从kernel中移除时，kobject core会自动发出KOBJ_REMOVE。

Reference counts

kobject的一个重要作用是给包含他的结构提供引用计数的功能。当对kobject有引用时，包含kobject的对象就应该存在。底层操作的函数时：

```
struct kobject *kobject_get(struct kobject *kobj);
void kobject_put(struct kobject *kobj);
```

kobject_put()会对object计数减一，如果需要，再释放kobject。
kobject_init()会将object的引用计数设置为1，因此上调用kobject_init的函数需要对kobject_put操作。

因为koject是动态的，所以kobject不能声明为静态变量或者在stack的变量，而是要动态的申请。未来的kernel版本会在运行时检查kobject是否动态创建。

如果你使用kobject仅仅是为了使用引用计数，那么需要使用kref而不是kobject.

Creating "simple" kobjects
有时候开发者仅仅是想要在sysfs中创建一个目录，不想要和kset, show, store这些很多很复杂的东西打交道。这里有一个例外可以创建独立的一个kobject：

```
struct kobject *kobject_create_and_add(char *name, struct kobject *parent);
```

查看samples/kobject/kobject-example.c和Documentation/filesystems/sysfs.txt来了解更多。

ktypes and release mothods
到目前为止，我们遗漏了一点，当kobject的引用计数减为0时，会发生什么？创建kobject的代码，往往不用关注发生了什么。

包含kobject的代码，只能创建kobject的代码，不能在kobject的引用计数减到0之前主动的去销毁kobject。引用计数也不是包含kobject的结构独立管理的。因此上，代码要具有异步通知的能力，这样可以在引用计数减为0后，进行销毁。

当kobject通过kobject_add()添加到系统中后，开发者不能直接调用kfree()。安全的办法是调用kobject_put()。在kobject_init()之后调用kobject_put()总是安全的办法。

异步通知是通过release()方法来完成的。通常release方法是

```
void my_object_release(struct kobject *kobj)
{
        struct my_object *mine = container_of(kobj, struct my_obj, kobj);
        /*.......*/
        kfree(mine);
}
```

有一点要反复强调，每一个kobject都必须有一个release方法，kobject必须在这个方法调用前都一直存在。如果这个条件不能满足，代码就是有缺陷的。如果没有release方法，kernel会发出警告。不要提供一个空的release方法。

kobject的名字在release的阶段还会存在，但是不能在这个回调函数中改名字。否则可能就会有内存泄露。

release方法不在kobject里，而是在他关联的ktype中。

```
struct kobj_type {
        void (*release)(struct kobject *kobj);
        const struct sysfs_ops *sysfs_ops;
        struct attribute **default_attrs;
        const struct kobj_ns_type_operations *(*child_ns_type)(struct kobject *kobj);
        const void *(*namespace)(struct kobject *kobj);
};
```

这个结构用来描述特定类型的kobject(或者更准确的说话是，描述包含kobject的结构的类型）。每一个kobject都要有关联的kobj_type，在kobject_init或者kobject_init_and_add()的时候要指定对应的kobj_type。

kobj_type中的release就是kobject的release方法。另外两个成员（sysfs_ops和default_attrs)表示该kobject如何在sysfs中进行控制。

default_attrs指向了默认的属性列表，当有kobject注册到该ktype时，这些属性就会默认的创建出来。

ksets

一个kset是一组彼此有联系的kobject。对于是否是同一个ktype没有要求，但是如果不是同一组ktype，需要小心处理。

kset有以下功能：

---是一组object的容器。kernel用kset来管理所有的block devices，或者所有的pci 设备驱动。

【？？？device是kobject，driver也是kobject】

---kset是sysfs中的子目录，通过kset自身的kobject表现出来。kset的kobject可以作为其他kobject的父object.

---kset也支持动态加载和删除kobjects，同时也会影响uevent来通知user space.

以面向对象的观点来看，kset就是一个顶层的容器类，kset有自己的kobject，但是这个kobject是kset自己来管理的，而不能是其他的。

kset用链表的方式管理子节点。【kset的子节点是啥？kobject还是kset?】kobject通过kset成员变量指向包含自己的kset。

kset由于有kobject，因此不能静态的或者在stack上创建。需要动态的创建：

```
struct kset *kset_create_and_add(const char *name, struct kset_uevent_ops *u, struct kobject *parent);
```

当使用完成后，调用

```
void kset_unregister(struct kset *kset);
```

来销毁。

查看samples/kobject/kset-example.c。

如果kset想要控制与kobject相关的**uevents**，那么设置kset_uevent_ops。

```
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);
};
```

filter函数允许kset阻止特定uevent传递到userspace。如果这个函数返回0，则改uevent就不会传递到userspace.
name函数修改uevent传递到userspace的默认的kset的名字。默认的，会传递kset自己的名字。
uevent函数是，uevent传递到userspace之前，如果有额外的环境变量需要传递，可以在这个函数中修改。

kobject通过kobject_add来添加到kset之前。kobject_add之前，要指定kobject的kset。

Kobject removal

调用kobject_put来减少引用计数，kobject core掉调用对应的ktype中的release来移除对应的Object。

循环引用，必须调用kobject_del来显示的打断循环引用。