**Linux设备模型(3)\_Uevent**

**1. Uevent的功能**

Uevent是Kobject的一部分，用于在Kobject状态发生改变时，例如增加、移除等，通知用户空间程序。用户空间程序收到这样的事件后，会做相应的处理。

该机制通常是用来支持热拔插设备的，例如U盘插入后，USB相关的驱动软件会动态创建用于表示该U盘的device结构（相应的也包括其中的kobject），并告知用户空间程序，为该U盘动态的创建/dev/目录下的设备节点，更进一步，可以通知其它的应用程序，将该U盘设备mount到系统中，从而动态的支持该设备。

**2. Uevent在kernel中的位置**

下面图片描述了Uevent模块在内核中的位置：

[](http://www.wowotech.net/content/uploadfile/201403/272229a8f5b5912c352e857e119e516320140311063951.gif)

由此可知，Uevent的机制是比较简单的，设备模型中任何设备有事件需要上报时，会触发Uevent提供的接口。Uevent模块准备好上报事件的格式后，可以通过两个途径把事件上报到用户空间：一种是通过kmod模块，直接调用用户空间的可执行文件；另一种是通过netlink通信机制，将事件从内核空间传递给用户空间。

注1：有关kmod和netlink，会在其它文章中描述，因此本文就不再详细说明了。

**3. Uevent的内部逻辑解析**

**3.1 Source Code位置**

Uevent的代码比较简单，主要涉及kobject.h和kobject\_uevent.c两个文件，如下：

* include/linux/kobject.h
* lib/kobject\_uevent.c

**3.2 数据结构描述**

kobject.h定义了uevent相关的常量和数据结构，如下：

* kobject\_action

1: /\* include/linux/kobject.h, line 50 \*/

2: enum kobject\_action {

3:     KOBJ\_ADD,

4:     KOBJ\_REMOVE,

5:     KOBJ\_CHANGE,

6:     KOBJ\_MOVE,

7:     KOBJ\_ONLINE,

8:     KOBJ\_OFFLINE,

9:     KOBJ\_MAX

10: };

kobject\_action定义了event的类型，包括：

ADD/REMOVE，Kobject（或上层数据结构）的添加/移除事件。

ONLINE/OFFLINE，Kobject（或上层数据结构）的上线/下线事件，其实是是否使能。

CHANGE，Kobject（或上层数据结构）的状态或者内容发生改变。

MOVE，Kobject（或上层数据结构）更改名称或者更改Parent（意味着在sysfs中更改了目录结构）。

CHANGE，如果设备驱动需要上报的事件不再上面事件的范围内，或者是自定义的事件，可以使用该event，并携带相应的参数。

* kobj\_uevent\_env

1: /\* include/linux/kobject.h, line 31 \*/

2: #define UEVENT\_NUM\_ENVP 32 /\* number of env pointers \*/

3: #define UEVENT\_BUFFER\_SIZE 2048 /\* buffer for the variables \*/

4:

5: /\* include/linux/kobject.h, line 116 \*/

6: struct kobj\_uevent\_env {

7:     char \*envp[UEVENT\_NUM\_ENVP];

8:     int envp\_idx;

9:     char buf[UEVENT\_BUFFER\_SIZE];

10:    int buflen;

11: };

前面有提到过，在利用Kmod向用户空间上报event事件时，会直接执行用户空间的可执行文件。而在Linux系统，可执行文件的执行，依赖于环境变量，因此kobj\_uevent\_env用于组织此次事件上报时的环境变量。

envp，指针数组，用于保存每个环境变量的地址，最多可支持的环境变量数量为UEVENT\_NUM\_ENVP。

envp\_idx，用于访问环境变量指针数组的index。

buf，保存环境变量的buffer，最大为UEVENT\_BUFFER\_SIZE。

buflen，访问buf的变量。

* kset\_uevent\_ops

1: /\* include/linux/kobject.h, line 123 \*/

2: struct kset\_uevent\_ops {

3:     int (\* const filter)(struct kset \*kset, struct kobject \*kobj);

4:     const char \*(\* const name)(struct kset \*kset, struct kobject \*kobj);

5:     int (\* const uevent)(struct kset \*kset, struct kobject \*kobj,

6:                         struct kobj\_uevent\_env \*env);

7: };

kset\_uevent\_ops是为kset量身订做的一个数据结构，里面包含filter和uevent两个回调函数，用处如下：

filter，当任何Kobject需要上报uevent时，它所属的kset可以通过该接口过滤，阻止不希望上报的event，从而达到从整体上管理的目的。

name，该接口可以返回kset的名称。如果一个kset没有合法的名称，则其下的所有Kobject将不允许上报uvent

uevent，当任何Kobject需要上报uevent时，它所属的kset可以通过该接口统一为这些event添加环境变量。因为很多时候上报uevent时的环境变量都是相同的，因此可以由kset统一处理，就不需要让每个Kobject独自添加了。

**3.3 内部动作**

通过kobject.h，uevent模块提供了如下的API（这些API的实现是在"lib/kobject\_uevent.c”文件中）：

1: /\* include/linux/kobject.h, line 206 \*/

2: int kobject\_uevent(struct kobject \*kobj, enum kobject\_action action);

3: int kobject\_uevent\_env(struct kobject \*kobj, enum kobject\_action action,

4:                         char \*envp[]);

5:

6: \_\_printf(2, 3)

7: int add\_uevent\_var(struct kobj\_uevent\_env \*env, const char \*format, ...);

8:

9: int kobject\_action\_type(const char \*buf, size\_t count,

10:                         enum kobject\_action \*type);

kobject\_uevent\_env，以envp为环境变量，上报一个指定action的uevent。环境变量的作用是为执行用户空间程序指定运行环境。具体动作如下：

* 查找kobj本身或者其parent是否从属于某个kset，如果不是，则报错返回（注2：由此可以说明，如果一个kobject没有加入kset，是不允许上报uevent的）
* 查看kobj->uevent\_suppress是否设置，如果设置，则忽略所有的uevent上报并返回（注3：由此可知，可以通过Kobject的uevent\_suppress标志，管控Kobject的uevent的上报）
* 如果所属的kset有uevent\_ops->filter函数，则调用该函数，过滤此次上报（注4：这佐证了3.2小节有关filter接口的说明，kset可以通过filter接口过滤不希望上报的event，从而达到整体的管理效果）
* 判断所属的kset是否有合法的名称（称作subsystem，和前期的内核版本有区别），否则不允许上报uevent
* 分配一个用于此次上报的、存储环境变量的buffer（结果保存在env指针中），并获得该Kobject在sysfs中路径信息（用户空间软件需要依据该路径信息在sysfs中访问它）
* 调用add\_uevent\_var接口（下面会介绍），将Action、路径信息、subsystem等信息，添加到env指针中
* 如果传入的envp不空，则解析传入的环境变量中，同样调用add\_uevent\_var接口，添加到env指针中
* 如果所属的kset存在uevent\_ops->uevent接口，调用该接口，添加kset统一的环境变量到env指针
* 根据ACTION的类型，设置kobj->state\_add\_uevent\_sent和kobj->state\_remove\_uevent\_sent变量，以记录正确的状态
* 调用add\_uevent\_var接口，添加格式为"SEQNUM=%llu”的序列号
* 如果定义了"CONFIG\_NET”，则使用netlink发送该uevent
* 以uevent\_helper、subsystem以及添加了标准环境变量（HOME=/，PATH=/sbin:/bin:/usr/sbin:/usr/bin）的env指针为参数，调用kmod模块提供的call\_usermodehelper函数，上报uevent。   
  其中uevent\_helper的内容是由内核配置项CONFIG\_UEVENT\_HELPER\_PATH(位于./drivers/base/Kconfig)决定的(可参考lib/kobject\_uevent.c, line 32)，该配置项指定了一个用户空间程序（或者脚本），用于解析上报的uevent，例如"/sbin/hotplug”。   
  call\_usermodehelper的作用，就是fork一个进程，以uevent为参数，执行uevent\_helper。

kobject\_uevent，和kobject\_uevent\_env功能一样，只是没有指定任何的环境变量。

add\_uevent\_var，以格式化字符的形式（类似printf、printk等），将环境变量copy到env指针中。

kobject\_action\_type，将enum kobject\_action类型的Action，转换为字符串。

|  |
| --- |
| **说明：怎么指定处理uevent的用户空间程序(简称uevent helper)？   上面介绍kobject\_uevent\_env的内部动作时，有提到，Uevent模块通过Kmod上报Uevent时，会通过call\_usermodehelper函数，调用用户空间的可执行文件（或者脚本，简称uevent helper ）处理该event。而该uevent helper的路径保存在uevent\_helper数组中。   可以在编译内核时，通过CONFIG\_UEVENT\_HELPER\_PATH配置项，静态指定uevent helper。但这种方式会为每个event fork一个进程，随着内核支持的设备数量的增多，这种方式在系统启动时将会是致命的（可以导致内存溢出等）。因此只有在早期的内核版本中会使用这种方式，现在内核不再推荐使用该方式。因此内核编译时，需要把该配置项留空。   在系统启动后，大部分的设备已经ready，可以根据需要，重新指定一个uevent helper，以便检测系统运行过程中的热拔插事件。这可以通过把helper的路径写入到"/sys/kernel/uevent\_helper”文件中实现。实际上，内核通过sysfs文件系统的形式，将uevent\_helper数组开放到用户空间，供用户空间程序修改访问，具体可参考"./kernel/ksysfs.c”中相应的代码，这里不再详细描述。** |