**Linux设备模型(7)\_Class**

**1. 概述**

在设备模型中，Bus、Device、Device driver等等，都比较好理解，因为它们对应了实实在在的东西，所有的逻辑都是围绕着这些实体展开的。而本文所要描述的Class就有些不同了，因为它是虚拟出来的，只是为了抽象设备的共性。

举个例子，一些年龄相仿、需要获取的知识相似的人，聚在一起学习，就构成了一个班级（Class）。这个班级可以有自己的名称（如295），但如果离开构成它的学生（device），它就没有任何存在意义。另外，班级存在的最大意义是什么呢？是由老师讲授的每一个课程！因为老师只需要讲一遍，一个班的学生都可以听到。不然的话（例如每个学生都在家学习），就要为每人请一个老师，讲授一遍。而讲的内容，大多是一样的，这就是极大的浪费。

设备模型中的Class所提供的功能也一样了，例如一些相似的device（学生），需要向用户空间提供相似的接口（课程），如果每个设备的驱动都实现一遍的话，就会导致内核有大量的冗余代码，这就是极大的浪费。所以，Class说了，我帮你们实现吧，你们会用就行了。

这就是设备模型中Class的功能，再结合内核的注释：A class is a higher-level view of a device that abstracts out low-level implementation details(include/linux/device.h line326)，就容易理解了。

**2. 数据结构描述**

**2.1 struct class**

struct class是class的抽象，它的定义如下：

1: /\* include/linux/device.h, line 332 \*/

2: struct class {

3: const char \*name;

4: struct module \*owner;

5:

6: struct class\_attribute \*class\_attrs;

7: struct device\_attribute \*dev\_attrs;

8: struct bin\_attribute \*dev\_bin\_attrs;

9: struct kobject \*dev\_kobj;

10:

11: int (\*dev\_uevent)(struct device \*dev, struct kobj\_uevent\_env \*env);

12: char \*(\*devnode)(struct device \*dev, umode\_t \*mode);

13:

14: void (\*class\_release)(struct class \*class);

15: void (\*dev\_release)(struct device \*dev);

16:

17: int (\*suspend)(struct device \*dev, pm\_message\_t state);

18: int (\*resume)(struct device \*dev);

19:

20: const struct kobj\_ns\_type\_operations \*ns\_type;

21: const void \*(\*namespace)(struct device \*dev);

22:

23: const struct dev\_pm\_ops \*pm;

24:

25: struct subsys\_private \*p;

26: };

其实struct class和struct bus很类似，解释如下：

name，class的名称，会在“/sys/class/”目录下体现。

class\_atrrs，该class的默认attribute，会在class注册到内核时，自动在“/sys/class/xxx\_class”下创建对应的attribute文件。

dev\_attrs，该class下每个设备的attribute，会在设备注册到内核时，自动在该设备的sysfs目录下创建对应的attribute文件。

dev\_bin\_attrs，类似dev\_attrs，只不过是二进制类型attribute。

dev\_kobj，表示该class下的设备在/sys/dev/下的目录，现在一般有char和block两个，如果dev\_kobj为NULL，则默认选择char。

dev\_uevent，当该class下有设备发生变化时，会调用class的uevent回调函数。

class\_release，用于release自身的回调函数。

dev\_release，用于release class内设备的回调函数。在device\_release接口中，会依次检查Device、Device Type以及Device所在的class，是否注册release接口，如果有则调用相应的release接口release设备指针。

p，和“[Linux设备模型(6)\_Bus](http://www.wowotech.net/linux_kenrel/bus.html)”中struct bus结构一样，不再说明。

**2.2 struct class\_interface**

struct class\_interface是这样的一个结构：它允许class driver在class下有设备添加或移除的时候，调用预先设置好的回调函数（add\_dev和remove\_dev）。那调用它们做什么呢？想做什么都行（例如修改设备的名称），由具体的class driver实现。

该结构的定义如下：

1: /\* include/linux/device.h, line 434 \*/

2: struct class\_interface {

3: struct list\_head node;

4: struct class \*class;

5:

6: int (\*add\_dev) (struct device \*, struct class\_interface \*);

7: void (\*remove\_dev) (struct device \*, struct class\_interface \*);

8: };

**3. 功能及内部逻辑解析**

**3.1 class的功能**

看完上面的东西，蜗蜗依旧糊里糊涂的，class到底提供了什么功能？怎么使用呢？让我们先看一下现有Linux系统中有关class的状况（这里以input class为例）：

root@android:/ # ls /sys/class/input/ -l                                                                                             
lrwxrwxrwx root     root              2014-04-23 03:39 event0 -> ../../devices/platform/i2c-gpio.17/i2c-17/17-0066/max77693-muic/input/input0/event0   
lrwxrwxrwx root     root              2014-04-23 03:39 event1 -> ../../devices/platform/gpio-keys.0/input/input1/event1   
lrwxrwxrwx root     root              2014-04-23 03:39 event10 -> ../../devices/virtual/input/input10/event10   
lrwxrwxrwx root     root              2014-04-23 03:39 event2 -> ../../devices/platform/s3c2440-i2c.3/i2c-3/3-0048/input/input2/event2   
…   
lrwxrwxrwx root     root              2014-04-23 03:39 event8 -> ../../devices/platform/soc-audio/sound/card0/input8/event8   
lrwxrwxrwx root     root              2014-04-23 03:39 event9 -> ../../devices/platform/i2c-gpio.8/i2c-8/8-0020/input/input9/event9   
lrwxrwxrwx root     root              2014-04-23 03:39 input0 -> ../../devices/platform/i2c-gpio.17/i2c-17/17-0066/max77693-muic/input/input0   
…   
lrwxrwxrwx root     root              2014-04-23 03:39 mice -> ../../devices/virtual/input/mice

[root@android:/](mailto:root@android:/) # ls /sys/devices/platform/s3c2440-i2c.3/i2c-3/3-0048/input/input2/event2/ -l

-r--r--r-- root     root         4096 2014-04-23 04:08 dev   
lrwxrwxrwx root     root              2014-04-23 04:08 device -> ../../input2   
drwxr-xr-x root     root              2014-04-23 04:08 power   
lrwxrwxrwx root     root              2014-04-23 04:08 subsystem -> ../../../../../../../../class/input   
-rw-r--r-- root     root         4096 2014-04-23 04:08 uevent

root@android:/ # ls /sys/devices/virtual/input/mice/ -l                                         
-r--r--r-- root     root         4096 2014-04-23 03:57 dev   
drwxr-xr-x root     root              2014-04-23 03:57 power   
lrwxrwxrwx root     root              2014-04-23 03:57 subsystem -> ../../../../class/input   
-rw-r--r-- root     root         4096 2014-04-23 03:57 uevent

看上面的例子，发现input class也没做什么实实在在的事儿，它（input class）的功能，仅仅是：

* 在/sys/class/目录下，创建一个本class的目录（input）
* 在本目录下，创建每一个属于该class的设备的符号链接（如，把“sys/devices/platform/s3c2440-i2c.3/i2c-3/3-0048/input/input2/event2”设备链接到”/sys/class/input/event2”），这样就可以在本class目录下，访问该设备的所有特性（即attribute）
* 另外，device在sysfs的目录下，也会创建一个subsystem的符号链接，链接到本class的目录

算了，我们还是先分析一下Class的核心逻辑都做了哪些事情，至于class到底有什么用处，可以在后续具体的子系统里面（如input子系统），更为细致的探讨。

**3.2 class的注册**

class的注册，是由\_\_class\_register接口（它的实现位于"drivers/base/class.c, line 609"）实现的，它的处理逻辑和bus的注册类似，主要包括：

* 为class结构中的struct subsys\_private类型的指针（cp）分配空间，并初始化其中的字段，包括cp->subsys.kobj.kset、cp->subsys.kobj.ktype等等
* 调用kset\_register，注册该class（回忆“[Linux设备模型(6)\_Bus](http://www.wowotech.net/linux_kenrel/bus.html)”中的描述，一个class就是一个子系统，因此注册class也是注册子系统）。该过程结束后，在/sys/class/目录下，就会创建对应该class（子系统）的目录
* 调用add\_class\_attrs接口，将class结构中class\_attrs指针所指向的attribute，添加到内核中。执行完后，在/sys/class/xxx\_class/目录下，就会看到这些attribute对应的文件

**3.3 device注册时，和class有关的动作**

在"[Linux设备模型(5)\_device和device driver](http://www.wowotech.net/linux_kenrel/device_and_driver.html)”中，我们有讲过struct device和struct device\_driver这两个数据结构，其中struct device结构会包含一个struct class指针（这从侧面说明了class是device的集合，甚至，class可以是device的driver）。当某个class driver向内核注册了一个class后，需要使用该class的device，通过把自身的class指针指向该class即可，剩下的事情，就由内核在注册device时处理了。

本节，我们讲一下在device注册时，和class有关的动作：

device的注册最终是由device\_add接口（drivers/base/core.c）实现了，该接口中和class有关的动作包括：

* 调用device\_add\_class\_symlinks接口，创建3.1小节描述的各种符号链接，即：在对应class的目录下，创建指向device的符号链接；在device的目录下，创建名称为subsystem、指向对应class目录的符号链接
* 调用device\_add\_attrs，添加由class指定的attributes（class->dev\_attrs）
* 如果存在对应该class的add\_dev回调函数，调用该回调函数

**4. 结束语**

其实在这篇文章结束后，蜗蜗依旧没有弄清楚class在内核到底是怎么使用的。不过没关系，在后续的子系统的分析中（如input子系统、RTC子系统等），我们会看到很多class的使用用例。到时候，再回过头总结，就会很清楚了。