Title: I2C 从设备驱动

Time：2016-2-23 11:05:44

3. i2c设备驱动

3.1. 驱动注册

i2c从设备的驱动注册，使用的是i2c-core.c提供的接口：i2c\_register\_driver；其调用如下：

i2c\_register\_driver --> driver\_register --> bus\_add\_driver；

对bus\_add\_driver进行分析：

**//驱动注册的准备工作：初始化链表什么的**

关于device\_driver数据结构的 struct driver\_private \*p

设备驱动模型是通过kobject对设备驱动进行层次管理的，因此device\_driver应该包含kobject成员，linux是将kobject包含在struct driver\_private中，再在device\_driver中包含struct driver\_private；我们可以理解driver\_private是device\_driver的私有数据，由内核进行操作。

struct driver\_private 是在驱动注册的开始，动态申请，并初始化的。

klist\_init(&priv->klist\_devices, NULL, NULL);

初始化设备链表，每一个与该驱动匹配的device都会添加到该链表下。

priv->kobj.kset = bus->p->drivers\_kset;

指定该驱动所属的kset；

kobject\_init\_and\_add

初始化kobject，并将kobject添加到其对应的kset集合中（即bus->p->drivers\_kset）。

该函数最终是调用kobject\_add\_internal将kobject添加到对应的kset中；需要主要的是，如果kobject的parent如果为NULL，在此会将其parent设置为所属kset集合的kobject：parent = kobject\_get(&kobj->kset->kobj)；

接下来是为kobject创建文件夹：create\_dir(kobj)；从而能从/sys/目录下显示。

driver\_attach，将驱动和设备进行绑定

将遍历总线上的设备链表，查找可以匹配的设备，并绑定。

driver\_attach --> bus\_for\_each\_dev(drv->bus, NULL, drv, \_\_driver\_attach);

将函数指针\_\_driver\_attach传入bus\_for\_each\_dev，将每个查找得到的device进行驱动匹配。（注1）

**注1：driver\_attach 中只有一句话**

**int driver\_attach(struct device\_driver \*drv)**

**{**

**return bus\_for\_each\_dev(drv->bus, NULL, drv, \_\_driver\_attach);**

**}**

bus\_for\_each\_dev：

遍历总线上的所有设备，因为总线上的设备都是bus->p->klist\_devices链表上的一个节点，因此该函数其实就是对链表的遍历，具体可以参考klist。

\_\_driver\_attach（源码位置drivers/base/dd.c）：

进行设备和驱动匹配，如果匹配成功，尝试进行绑定。

**//驱动和设备匹配**

1. 首先进行匹配确认：driver\_match\_device(drv, dev)；

调用关系： --> drv->bus->match --> i2c\_device\_match （注1）--> of\_driver\_match\_device （注2）；i2c\_match\_id

**注1： 1. device\_driver与i2c\_device\_match建立关系：i2c\_register\_driver：driver->driver.bus = &i2c\_bus\_type**

**2. i2c\_bus\_type在I2C-core.c中定义并赋值：**

**struct bus\_type i2c\_bus\_type = {**

**.name = "i2c",**

**.dev\_attrs = i2c\_dev\_attrs,**

**.match = i2c\_device\_match,**

**.uevent = i2c\_device\_uevent,**

**.probe = i2c\_device\_probe,**

**.remove = i2c\_device\_remove,**

**.shutdown = i2c\_device\_shutdown,**

**.suspend = i2c\_device\_suspend,**

**.resume = i2c\_device\_resume,**

**};**

**注2：linux 2.6.32中没有找到该接口，在Linux3.4.12中找到**

可以看出，最终有两种方式进行驱动匹配查询：

方法一：通过of\_driver\_match\_device对比of\_device\_id；

方法二：通过i2c\_match\_id对比id\_table；

方法二实际上就是对比

i2c\_driver->id\_table->name 和client->name是否一致。

2. 如果匹配确认，进行驱动与设备绑定：driver\_probe\_device；

调用关系： driver\_probe\_device --> really\_probe

--> dev->bus->probe

driver\_bound

在really\_probe中，首先将设备的驱动指针指向该驱动：dev->driver = drv。

对应于i2c\_bus\_type，dev->bus->probe 即是：i2c\_device\_probe，最终调用驱动的probe函数。（注3）

**注3：**

**1. dev->bus->probe与i2c\_device\_probe的关系建立：i2c-core.c🡪i2c\_attach\_client—> client->dev.bus = &i2c\_bus\_type;**

**2. i2c\_device\_probe最终调用驱动的probe函数**

**i2c\_device\_probe（struct device \*dev）{**

**struct i2c\_driver \*driver = to\_i2c\_driver(dev->driver); *//relly\_probe中 dev->driver = drv***

**…**

**…**

**status = driver->probe(client, i2c\_match\_id(driver->id\_table, client)); //驱动的probe函数**

**}**

最后是driver\_bound，将驱动与设备进行绑定： //driver\_bound Linux2.6.32中无该接口 应该是后来版本新增

其实就是调用klist\_add\_tail：将设备节点添加到驱动的klist\_devices;

调用klist\_add\_tail，将被注册的驱动添加到总线的klist\_drivers上；

klist\_add\_tail(&priv->knode\_bus, &bus->p->klist\_drivers);

module\_add\_driver(drv->owner, drv)

在sysfs创建drivers目录

总结：i2c设备驱动注册过程

1. xxx\_init-->i2c\_add\_driver-->i2c\_register\_driver-->driver\_regster(&driver->driver)-->bus\_add\_driver-->driver\_attach-->bus\_for\_each\_dev-->\_\_driver\_attach-->driver\_match\_device-->drv->bus->match--> i2c\_device\_match--> i2c\_match\_id
2. \_\_driver\_attach 调用成功后：

driver\_probe\_device-->really\_probe-->dev->bus->probe-->i2c\_device\_probe-->driver->probe

1. 调用klist\_add\_tail，将被注册的驱动添加到总线的klist\_drivers上；

klist\_add\_tail(&priv->knode\_bus, &bus->p->klist\_drivers);