device_register分析

原创

chihunqi5879 于 2018-04-20 11:10:32 发布 1457 收藏 5

分类专栏： 驱动 安卓

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

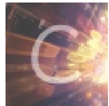
本文链接：https://blog.csdn.net/chihunqi5879/article/details/80015135

版权

驱动 同时被 2 个专栏收录

9 篇文章 2 订阅

订阅专栏


安卓

5 篇文章 0 订阅

订阅专栏

上篇文章分析了driver_register函数，这篇文章主要介绍device_register。内核在调用device_register的时候也会匹配已经加载好的设备驱动程序，从而执行probe函数。在i2c-core.c中i2c_new_device函数会调用device_register函数，下面开始分析device_register的源码(driver/base/core.c)：

```
int device_register(struct device *dev)
{
/*初始化dev，为了在sys下生成节点，会先初始化kobj和kset*/
    device_initialize(dev);
    return device_add(dev);//下面看device_add源码分析
}
```

device_add源码：

```
int device_add(struct device *dev)
{
    struct device *parent = NULL;
    struct kobject *kobj;
    struct class_interface *class_intf;
    int error = -EINVAL;

    dev = get_device(dev);//主要是获得kobj
    if (!dev)
        goto done;

    if (!dev->p) {
        error = device_private_init(dev);
        if (error)
            goto done;
    }
    if (dev->init_name) {
        dev_set_name(dev, "%s", dev->init_name);
        dev->init_name = NULL;
    }
```

```
    /* subsystems can specify simple device enumeration */
    if (!dev_name(dev) && dev->bus && dev->bus->dev_name)
        dev_set_name(dev, "%s%u", dev->bus->dev_name, dev->id);

    if (!dev_name(dev)) {
        error = -EINVAL;
        goto name_error;
    }

    pr_debug("device: '%s': %s\n", dev_name(dev), __func__);

    parent = get_device(dev->parent);
    kobj = get_device_parent(dev, parent);//得到设备
    if (kobj)
        dev->kobj.parent = kobj;//将辐射的kobj赋给dev

    /* use parent numa_node */
    if (parent)
        set_dev_node(dev, dev_to_node(parent));

    /* first, register with generic layer. */
    /* we require the name to be set before, and pass NULL */
    error = kobject_add(&dev->kobj, dev->kobj.parent, NULL);//将kobj加入到sys层次中，在sys/bus/i2c/device
    if (error)
        goto Error;

    /* notify platform of device entry */
    if (platform_notify)
        platform_notify(dev);
/*在sys/devices下的具体的设备目录下添加uevent属性*/
    error = device_create_file(dev, &uevent_attr);
    if (error)
        goto attrError;

    if (MAJOR(dev->devt)) {
    /*在sys/devices下的具体的设备目录下添加dev属性*/
        error = device_create_file(dev, &devt_attr);
        if (error)
            goto ueventattrError;
/*在/sys/dev/char/或者/sys/dev/block/创建devt的属性的连接文件(char是块设备，char时字符设备),形如10:45,由主设备
        error = device_create_sys_dev_entry(dev);
        if (error)
            goto devtattrError;

        devtmpfs_create_node(dev);
    }

    error = device_add_class_symlinks(dev);//穿件device下的具体设备与class之间的链接文件
    if (error)
        goto SymlinkError;
    error = device_add_attrs(dev);//添加设备属性文件
    if (error)
        goto AttrsError;
    error = bus_add_device(dev);//将设备添加到bus上，创建subsystem链接文
    件，链接class下的具体的子系统文件夹
    if (error)
        goto BusError;
    error = dpm_sysfs_add(dev);//添加设备的电源管理属性，截止这里，
    在devices具体目录下生成有以下四个属性文件：uevent，dev，subsystem,power,
    你找到了吗？
```

```
    你找到了吗?
    if (error)
        goto DPMError;
    device_pm_add(dev);//添加设备到激活设备列表中，用于电源管理
if (dev->bus)
        blocking_notifier_call_chain(&dev->bus->p->bus_notifier,
                         BUS_NOTIFY_ADD_DEVICE, dev);

    kobject_uevent(&dev->kobj, KOBJ_ADD);
    bus_probe_device(dev);//去bus上找dev对应的drv,主要执行
    __device_attach,主要进行match,sys_add,执行probe函数和绑定等操作
    if (parent)
        klist_add_tail(&dev->p->knode_parent,
                   &parent->p->klist_children);

    if (dev->class) {//如果该dev有所属类，则将dev的添加到类的设备列表里面
        mutex_lock(&dev->class->p->mutex);
        /* tie the class to the device */
        klist_add_tail(&dev->knode_class,
                   &dev->class->p->klist_devices);

        /* notify any interfaces that the device is here */
        list_for_each_entry(class_intf,
                    &dev->class->p->interfaces, node)
            if (class_intf->add_dev)
                class_intf->add_dev(dev, class_intf);
        mutex_unlock(&dev->class->p->mutex);
    }
done:
    put_device(dev);
    return error;
 DPMError:
    bus_remove_device(dev);
 BusError:
    device_remove_attrs(dev);
 AttrsError:
    device_remove_class_symlinks(dev);
 SymlinkError:
    if (MAJOR(dev->devt))
        devtmpfs_delete_node(dev);
    if (MAJOR(dev->devt))
        device_remove_sys_dev_entry(dev);
 devtattrError:
    if (MAJOR(dev->devt))
        device_remove_file(dev, &devt_attr);
 ueventattrError:
    device_remove_file(dev, &uevent_attr);
 attrError:
    kobject_uevent(&dev->kobj, KOBJ_REMOVE);
    kobject_del(&dev->kobj);
 Error:
    cleanup_device_parent(dev);
    if (parent)
        put_device(parent);
name_error:
    kfree(dev->p);
    dev->p = NULL;
    goto done;
}
```

device和driver的匹配过程是在函数bus_probe_device中完成的，下面看bus_probe_device源码（driver/base/bus.c）：

```
void bus_probe_device(struct device *dev)
{
    struct bus_type *bus = dev->bus;
    struct subsys_interface *sif;
    int ret;

    if (!bus)
        return;

    if (bus->p->drivers_autoprobe) {//如果需要自动匹配驱动
        ret = device_attach(dev);//进行设备和驱动的匹配
        WARN_ON(ret < 0);
    }

    mutex_lock(&bus->p->mutex);
    list_for_each_entry(sif, &bus->p->interfaces, node)
        if (sif->add_dev)
            sif->add_dev(dev, sif);
    mutex_unlock(&bus->p->mutex);
}
```

下面看device_attach源码。看其device是怎么和driver匹配的(driver/base/dd.c)：

```
int device_attach(struct device *dev)
{
    int ret = 0;

    device_lock(dev);
    if (dev->driver) {
        if (klist_node_attached(&dev->p->knode_driver)) {
            ret = 1;
            goto out_unlock;
        }
        ret = device_bind_driver(dev);//如果dev有dri则将设备和驱动进行绑定
        if (ret == 0)
            ret = 1;
        else {
            dev->driver = NULL;
            ret = 0;
        }
    } else {
        pm_runtime_get_noresume(dev);
        ret = bus_for_each_drv(dev->bus, NULL, dev, __device_attach);
        /*否则在总线上寻找驱动和设备的匹配，__device_attach是在一个函数指针
        和driver中分析的一样，先取出driver，再将dev和drv一同传入__device_attach*/
        pm_runtime_put_sync(dev);
    }
out_unlock:
    device_unlock(dev);
    return ret;
}
```

先看bus_for_each_drv（driver/base/bus.c）源码：

```
int bus_for_each_drv(struct bus_type *bus, struct device_driver *start,
             void *data, int (*fn)(struct device_driver *, void *))
{

    struct klist_iter i;
    struct device_driver *drv;
    int error = 0;

    if (!bus)
        return -EINVAL;

    klist_iter_init_node(&bus->p->klist_drivers, &i,
                 start ? &start->p->knode_bus : NULL);//遍历链表上多有的driver
    while ((drv = next_driver(&i)) && !error)
    /*next_driver函数就不往里跟了，这个函数就是循环取出每一个driver*/
        error = fn(drv, data);
    /*将取出的drv和dev传入fn中，即__device_attach函数*/
    klist_iter_exit(&i);
    return error;
}
```

下面看__device_attach(driver/base/dd.c)函数源码：

```
static int __device_attach(struct device_driver *drv, void *data)
{
    struct device *dev = data;

    if (!driver_match_device(drv, dev))//先进性dev和drv的匹配
        return 0;

    return driver_probe_device(drv, dev);//匹配成功后，执行此函数，跟进去就是
    执行driver的probe函数，这个函数在driver_register文章中跟过了*/
}
```

driver_match_device函数和driver_probe_device函数在上一篇driver_register中分析过了，driver_match_device函数会自动调用bus总线的注册过的match函数，driver_probe_device函数是最终会先执行bus的probe函数 ，在执行driver的probe函数。
分析到这，再往下分析的过程就是和driver_register中一样的（连接下来要执行的函数都是一样的），先是drv和dev匹配，再执行drv的probe函数。
从而可以看出来，无论是先注册device还是先注册driver，其都会去匹配相应的设备和驱动。