复习总线设备驱动模型，做了一点小笔记，大牛略过。

**一、Linux系统的驱动框架的基础很大一部分是围绕着总线设备驱动模型展开的。**

**二、涉及3个重要结构体：**

struct bus\_type：总线

struct device ：设备

struct device\_driver：驱动

**三、结构体核心代码分析(2.6.38内核)**

struct bus\_type

{

    const char       \*name; // 总线名

    struct bus\_attribute   \*bus\_attrs; //总线属性文件，会显示在/sys/xxx中

    struct device\_attribute   \*dev\_attrs;  //设备属性文件

    struct driver\_attribute   \*drv\_attrs;   // 驱动属性文件

    int (\*match)(struct device\*dev, struct device\_driver \*drv);// 驱动与设备是否匹配的检测函数

    struct subsys\_private \*p;

};

其中，struct subsys\_private包含一个设备链表(struct klist klist\_devices)和一个驱动链表(   struct klist klist\_drivers)

struct device

{

    struct kobject kobj;

    const char                           //设备名

    struct bus\_type   \*bus;        // 该设备挂接在哪条总线上

    struct device\_driver\*driver; //该设备所对应的驱动

    void   \*platform\_data;    // 平台特定数据，一般我们移植内核时需填充该结构体(如支持mini2440的nandflash，dm9000等)

    dev\_t           devt;             // 设备号

  };

任何建立在平台总线设备驱动模型基础上的驱动代码(如平台驱动，PCI驱动，USB驱动，I2C驱动，SPI驱动等)，它们的设备结构体(如platform\_device, pci\_dev，usb\_device，i2c\_device, spi\_device等)都包含一个struct device结构体，当这些驱动向内核注册各式各样的设备时，其实最终都会调用到：

int device\_register(struct device \*dev)

{

    device\_initialize(dev);  //做各类初始化

    //将设备挂接在对应的总线上，主要工作把设备(device)添加到总线       (bus\_type)的klist\_devices链表

    return device\_add(dev);

}

struct device\_driver {

    const char       \*name;// 驱动名

    struct bus\_type       \*bus;// 该驱动所属的总线

    int (\*probe) (struct device \*dev);// 指向设备探测函数

    int (\*remove) (struct device\*dev);// 指向设备移除函数

    struct driver\_private \*p;

 };

同理，任何建立在平台总线设备驱动模型基础上的驱动代码(如平台驱动，PCI驱动，USB驱动，I2C驱动，SPI驱动等)，它们的驱动结构体(如platform\_driver, pci\_driver，usb\_driver,i2c\_driver, spi\_driver等)都包含一个struct device\_driver结构体，当这些驱动向内核注册各式各样的驱动时，最终都会调用到：

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

{

    // 将驱动绑定在对应的总线上，主要工作把驱动(device\_driver)添加到总线(bus\_type)的klist\_drivers链表中去

     ret = bus\_add\_driver(drv);

}

无论是调用driver\_register()注册驱动, 还是用device\_register注册设备, 内核都会调用总线的match函数来探测是否有合适device\_driver的device或者是否有合适device的device\_driver，如果match成功，则会调用device\_driver的probe函数进行更进一步的探测。

这样我们就可以站在一个新的高度上看驱动了

更具体的内容比如怎么探测设置probe请看韦东山2期驱动大全相关视频。