回顾：
1.linux内核platform机制
  目的：将硬件和软件分离
  一个虚拟总线：platform_bus_type
  两个链表：dev链表(硬件)和drv链表(软件)
  两个数据结构：
      struct platform_device
              .name
              .id
              .resource
              .num_resources
              .dev
                      .release
      struct platform_driver
              .driver
                      .name
              .probe
                      1.获取硬件信息
                              platform_get_resource
                      2.处理硬件信息
                        4个该
                      3.注册硬件操作接口
              .remove
                      对着干
   四个配套函数
   platform_device_register
   platform_device_unregister
   platform_driver_register
   platform_driver_unregister
   四个什么都是由内核来完成

   一个关心：probe函数是否被调用

2.I2C总线
  面试题：谈谈对I2C总线的理解
  2.1.I2C总线功能
  2.2.I2C概念
  2.3.三个问题
  2.4.协议设计概念
      START
      STOP
      设备地址
      读写位
      ACK
  2.5.访问过程
      举例子：以CPU向AT24C02片内地址0x10存储空间写入字符'A'为例
      结论：一切的操作都是在芯片手册的操作时序图中
  2.6.配合
      画出时序图
      START 设备地址=0x50 W ACK

3.linux内核I2C驱动开发
  3.1.明确I2C总线实际的硬件操作
  研究对象转移：
  CPU访问外设
  转移到CPU访问SDA和SCL

转移到CPU访问I2C控制器
转移到CPU访问I2C控制器对应的寄存器

3.2.linux内核I2C驱动的分类
I2C总线驱动：
管理的硬件是I2C控制器
此驱动操作I2C控制器最终帮你发起SCL和SDA的时序
注意：像START,ACK,读写位,STOP这些都是标准信号,I2C控制器
自动完成,但是设备地址,操作的芯片的片内地址和片内数据
控制器是不知道的,这些由I2C设备驱动来告诉
I2C总线驱动
总线驱动由CPU芯片厂家实现好,只需配置内核支持即可：

```
cd /opt/kernel
make menuconfig
  Device Drivers->
        I2C supports->
                i2c hardware bus supports...-> //在此
                        <*> s3c2410 i2c ...
```

I2C设备驱动：
管理的硬件是I2C外设本身
像设备地址,片内地址,片内数据都是跟
外设相关,I2C设备驱动需要将这些数据
信息丢给I2C总线驱动,最终完成硬件SCL
和SDA的时序传输
I2C驱动开发的重点！

问：以上两个驱动如何关联呢？

3.3.linux内核I2C驱动框架(分层思想)
例如：以CPU向AT24C02片内地址0x10存储空间写入数据'A'为例：

应用层：
作用：就是要获取到底层硬件将来要操作的数据信息(片内地址和片内数据)

```
struct at24c02_info {
        unsigned char addr;
        unsigned char data;
};
struct at24c02_info at24c02;
at24c02.addr = 0x10;
at24c02.data = 'A';
//将应用层的数据丢给I2C设备驱动
ioctl(fd, AT24C02_WRITE, &at24c02);
```

I2C设备驱动层：
从用户获取要操作的数据信息或者将数据信息丢给用户
同时还要将数据信息利用SMBUS接口丢给I2C总线驱动
或者利用SMBUS接口从I2C总线驱动获取数据

```
at24c02_ioctl(file, cmd, arg) {
   struct at24c02_info at24c02;
   copy_from_user
       结果：
       at24c02.addr = 0x10
       at24c02.data = 'A'
   switch ....
       case AT24C02_WRITE: //写
            I2C设备驱动利用内核提供的SMBUS接口函数
```

将这些数据信息丢给I2C总线驱动：
smbus_xxxx(数据信息);
break

}

SMBUS接口层：
连接I2C设备驱动和I2C总线驱动
桥梁作用
内核已经实现！
smbus_xxx
0x10
'A'

I2C总线驱动层：
唯一的作用就是操作I2C控制器,启动硬件的最终传输
设备地址同样由I2C设备驱动来告诉
当然传输的数据信息来自用户0x10,'A'

硬件层：
START->设备地址|0->ACK->0x10->ACK->'A'->ACK->STOP

问：linux内核I2C设备驱动如何编写呢？
答：同样采用分离思想(bus-device-drivers编程模型)
1.首先内核已经定义好了一个虚拟总线叫i2c_bus_type
在这个总线上维护着两个链表dev链表和drv链表
2.dev链表上每一个节点描述的I2C外设的硬件信息,对应的
数据结构为struct i2c_client, 每当添加一个I2C外设的硬件信息时,
只需利用此数据结构定义初始化一个对象,添加到dev链表以后
内核会帮你遍历drv链表,取出drv链表上每一个软件软件跟这个
硬件节点进行匹配(匹配通过内核调用总线提供的match函数,比较
i2c_client的name和i2c_driver的id_table的name)如果匹配成功
内核调用软件节点的probe函数,并且把匹配成功的硬件节点的
首地址给probe函数
3.drv链表上每一个节点描述的I2C外设的软件信息,对应的
数据结构为struct i2c_driver, 每当添加一个I2C外设的软件信息时,
只需利用此数据结构定义初始化一个对象,添加到drv链表以后
内核会帮你遍历dev链表,取出dev链表上每一个硬件节点跟这个
软件节点进行匹配(匹配通过内核调用总线提供的match函数,比较
i2c_client的name和i2c_driver的id_table的name)如果匹配成功
内核调用软件节点的probe函数,并且把匹配成功的硬件节点的
首地址给probe函数

总结：如果要实现一个I2C外设的I2C设备驱动只需关注两个数据结构：
struct i2c_client
struct i2c_driver

3.4.struct i2c_client
struct i2c_client {
unsigned short addr;//设备地址,将来寻找外设
char name[I2C_NAME_SIZE];//将来用于匹配
int irq; //中断号
...
};
功能：描述I2C外设的硬件信息
切记：addr,name两个成员必须要进行初始化！

结论：驱动开发者不会自己去拿这个结构体去定义初始化和注册
一个硬件节点对象到内核,而是利用以下结构体间接完成
以上操作(用i2c_client定义初始化注册对象)

神秘的数据机构：
```
struct i2c_board_info {
    char type[I2C_NAME_SIZE];
    unsigned short  addr;
    int irq;
    ...
};
```
功能：将来驱动开发者用此数据结构去定义初始化注册
一个I2C外设的硬件信息到内核,将来内核会根据
你提供的硬件信息,内核会帮你定义初始化和注册
一个i2c_client硬件节点对象到内核dev链表中
成员：
type:硬件信息的名称,将来这个字段的内容会赋值给
i2c_client的name成员,用于匹配
addr:I2C外设的设备地址,将来这个字段的内容会赋值给
i2c_client的addr成员,用于找某个外设
irq:I2C外设的中断号,将来这个字段的内容会赋值给
i2c_client的irq成员
切记：type,addr必须要初始化！
内核鼓励使用I2C_BOARD_INFO宏对type,addr进行初始化！
例如：I2C_BOARD_INFO("at24c02", 0x50)

配套函数：
注册i2c_board_info描述的硬件信息到内核的函数：
```
int i2c_register_board_info(int busnum,
                    const struct i2c_board_info *info,
                            int number)
```
功能:i2c_board_info描述的硬件信息到内核
参数：
busnum:I2C外设所在的I2C总线编号(从0开始),通过原理图来获取
info:传递定义初始化的I2C外设的硬件信息对象首地址
number:硬件信息的个数
结果：将来内核根据你注册的硬件信息,内核会帮你定义初始化
和注册一个i2c_client对象到内核中
切记切记：i2c_board_info定义初始化和注册不能采用insmod
和rmmod,此代码必须放在平台代码中完成！
TPAD的平台代码：arch/arm/mach-s5pv210/mach-cw210.c

案例：向内核添加AT24C02的硬件节点到内核
实施步骤：
1.cd /opt/kernel
2.vim arch/arm/mach-s5pv210/mach-cw210.c 在头文件的后面添加如下内容：
```
  //定义初始化AT24C02的硬件信息
  static struct i2c_board_info at24c02[] = {
      {
            I2C_BOARD_INFO("at24c02", 0x50)
      }
  };//"at24c02"将来用于匹配,0x50表示设备地址,用于寻找外设
```
3.vim arch/arm/mach-s5pv210/mach-cw210.c 在任何一个函数中调用
一下函数完成对I2C外设硬件信息的注册,推荐找到：
```
 .init_machine      = smdkc110_machine_init,
```

找到smdkc110_machine_init,在此函数中调用：
i2c_register_board_info(0, at24c02, ARRAY_SIZE(at24c02));
保存退出

4.make zImage
cp arch/arm/boot/zImage /tftpboot

5.用新内核重启开发板
至此内核就有了AT24C02的硬件信息,内核也帮你定义初始化
注册一个i2c_client,此时此刻它会静静等待着软件节点的到来!

3.5.struct i2c_driver

```
struct i2c_driver {
    int (*probe)(struct i2c_client *client,
                 const struct i2c_device_id *id);
    int (*remove)(struct i2c_client *client);
    const struct i2c_device_id *id_table;
    ...
};
```

作用：描述I2C外设的软件信息
id_table:I2C外设的标识,其中的name用于匹配

```
struct i2c_device_id {
        char name[I2C_NAME_SIZE];//用于匹配
        unsigned long data; //给probe函数传递的参数
};
例如：
static const struct i2c_device_id at24_id[] = {
        { "at24c02", 0 },
        { }
};//"at24c02"将来用于匹配
```

probe:匹配成功,内核调用此函数
形参client指向匹配成功的硬件信息
id指向驱动自己定义初始化的i2c_device_id对象,就是at24_id

remove：卸载软件节点,内核调用此函数

配套函数：

```
i2c_add_driver(&软件节点对象); //注册
i2c_del_drier(&软件节点对象);  //卸载
```

案例：编写TPAD上AT24C02存储器的软件节点代码
注意：前提是内核已经有了AT24C02的硬件信息
实施步骤：
1.从ftp下载源码
2.mkdir /opt/drivers/day12/
cp at24c02_1 /opt/drivers/day12
cd /opt/drivers/day12/at24c02_1 潜心研究代码
make
cp at24c02_drv.ko /opt/rootfs/home/drivers
开发板测试：
insmod /home/drivers/at24c02_drv.ko
观察驱动的probe函数是否被调用

rmmod at24c02_drv //观察remove函数是否被调用

案例：编写TPAD上AT24C02存储器的软件节点代码
注意：前提是内核已经有了AT24C02的硬件信息

实施步骤：
1.从ftp下载源码
2.mkdir /opt/drivers/day12/
  cp at24c02_2 /opt/drivers/day12
  cd /opt/drivers/day12/at24c02_2
  潜心研究at24c02_drv.c at24c02_test.c
  make
  cp at24c02_drv.ko /opt/rootfs/home/drivers
  arm-linux-gcc -o at24c02_test at24c02_test.c
  cp at24c02_test /opt/rootfs/home/drivers/

开发板测试：
insmod /home/drivers/at24c02_drv.ko
ls /dev/at24c02 -lh
./at24c02_test

SMBUS接口函数的使用操作步骤：
1.首先打开I2C外设的芯片手册,找到对应的操作时序图
2.然后打开内核关于smbus接口的说明文档：
  内核源码\Documentation\i2c\smbus-protocol
3.在文档中根据手册的时序要求找到对应的函数
  例如：
  按字节写的时序：S->设备地址<<1|0->ACK->片内地址->ACK->数据->ACK->STOP
  找到此函数：i2c_smbus_write_byte_data()
  此函数说明提示：S Addr Wr [A] Comm [A] Data [A] P //完全符合
4.赋值函数名,在sourceinsight中找到函数的定义
  在自己的驱动代码中填充参数即可