1. 什么是平台总线模型？

平台总线模型又称 platform 总线模型。（这个总线是虚拟出来的总线，不是真实存在的）是Linux内核虚拟出来的一条总线，并不是真实的导线。

1. 为什么会有平台总线模型？

我们前几节视频，讲解了字符设备驱动 和 杂项设备驱动。我们通过对比字符设备的.c文件 和 杂项设备的.c文件。可以看出其代码量是极大的。

但是这两个驱动都有一个共同的特点：都把驱动 与 设备写到了一个.c文件里，比如字符设备驱动的.c文件里面：

驱动部分的代码都写在了 init函数里面。

与设备有关的代码都写在了file\_operation对应的几个函数里面的。如：open、write。我们可以在open里面对管脚进行初始化，或者对寄存器进行一些操作。

虽然这么做十分的简单，但是当驱动变多之后，就会出现许多的问题。如：

现在有一个硬件平台，硬件平台上有很多的模块，假如有500个模块。而这些模块都用到了LED灯，如果使用杂项设备来写，虽然相比于字符设备代码要少，但也要写500份这样的代码。然后生成设备节点，供上层应用 去控制在不同模块下的LED灯。这样就带来了两个大问题：

1. 写了大量重复性的代码。
2. 重用性不高。

而platform可以提高代码的重用性，以及减少重复性的代码。

它之所以可以做到这两点，是因为平台总线模型 把 驱动 和 硬件 分开了：

由原来的:

设备 <----> 驱动

变成了：

设备 <----> 总线 <----> 驱动

device.c driver.c

分成两个.c文件之后，那么：

所有设备的代码都放到device.c文件里面。也就是说LED灯用到的所有管脚，都在device.c里面进行描述。

所有驱动的代码都放到driver.c文件里面。那么这500个LED灯的驱动，其中的重复性代码，只需要写一遍，只写成一个driver.c就可以了。

当涉及平台移植的时候，driver.c就不需要进行修改。因为driver.c里面都是一些稳定不变的代码。不论放到那个开发板都是通用的。

总结：

平台总线模型 就是 把原来的驱动.c文件 分成了 两个.c文件，一个是device.c与设备相关的，另一个是driver.c与驱动相关的。

把稳定不变的放到driver.c文件里面。把需要修改改变的放在了device.c文件里面。

1. 平台总线模型的优点？
2. 驱动和设备的代码是分开的，可以减少重复性代码
3. 都挂载到了总线上，方便进行管理。
4. 怎么编写以平台总线模型设计的驱动？
5. 分别写driver.c和device.c文件。然后分别注册。
6. 写好之后，当使用insmod 加载的时候，就会去检测有没有一样的driver.c/device.c。如果有一样的，他就会去进行匹配。(也就是说，总线如果检测到有一样的driver.c/device.c。就会将二者进行匹配。)
7. 平台总线是如何判断有没有匹配的呢？

平台总线是以名字来匹配的。实质上就是字符串比较。（由于是把原来的一个.c分成了两个，所以在用的时候，就需要合成一个。就需要总线进行名字匹配（牵手）。）