Platform

平台设备和驱动
~~~~~~~~~~~~~
在<linux/platform_device.h>中可以找到面向平台总线的驱动模型接口: platform_device和
platform_driver. 平台总是条伪总线,它被用来连接处在仅有最少基本组件的总线上的那些设备.
这样的总线包括许多片上系统上的那些用来整合外设的总线, 也包括一些"古董"PC上的连接器; 但不包
括像PCI或USB这样的有庞大正规说明的总线.

平台设备
~~~~~~
平台设备通常指的是系统中的自治体, 包括老式的基于端口的设备和连接外设总线的北桥(host bridges),
以及集成在片上系统中的绝大多数控制器. 它们通常拥有的一个共同特征是直接编址于CPU总线上. 即使在某
些罕见的情况下, 平台设备会通过某段其他类型的总线连入系统, 它们的寄存器也会被直接编址.

平台设备会分到一个名称(用在驱动绑定中)以及一系列诸如地址和中断请求号(IRQ)之类的资源.

```
struct platform_device {
        const char      *name;
        u32             id;
        struct device   dev;
        u32             num_resources;
        struct resource *resource;
};
```

平台驱动
~~~~~~
平台驱动遵循标准驱动模型的规范, 也就是说发现/列举(discovery/enumeration)在驱动之外处理, 而由驱动提供probe()和remove方法. 平台驱动按标准规范对电源管理和关机通告提供支持.

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        void (*shutdown)(struct platform_device *);
        int (*suspend)(struct platform_device *, pm_message_t state);
        int (*suspend_late)(struct platform_device *, pm_message_t state);
        int (*resume_early)(struct platform_device *);
        int (*resume)(struct platform_device *);
        struct device_driver driver;
};
```

注意probe()总应该核实指定的设备硬件确实存在;平台设置代码有时不能确定这一点. 枚举(probing)

可以使用的设备资源包括时钟及设备的platform_data.
(译注: platform_data定义在device.txt中的"基本设备结构体"中.)

平台驱动通过普通的方法注册自身:

```
        int platform_driver_register(struct platform_driver *drv);
```

或者, 更常见的情况是已知设备不可热插拔, probe()过程便可以驻留在一个初始化区域(init section)中,以便减少驱动的运行时内存占用(memory footprint):

```
        int platform_driver_probe(struct platform_driver *drv,
                                  int (*probe)(struct platform_device *))
```

设备列举
~~~~~~~

按规定, 应由针对平台(也适用于针对板)的设置代码来注册平台设备:

```
int platform_device_register(struct platform_device *pdev);

int platform_add_devices(struct platform_device **pdevs, int ndev);
```

一般的规则是只注册那些实际存在的设备, 但也有例外. 例如, 某外部网卡未必会装配在所有的板子上,
或者某集成控制器所在的板上可能没挂任何外设, 而内核却需要被配置来支持这些网卡和控制器.

有些情况下, 启动固件(boot firmware)会导出一张装配到板上的设备的描述表. 如果没有这张表, 通常
就只能通过编译针对目标板的内核来让系统设置代码安装正确的设备了. 这种针对板的内核在嵌入式和自定
义的系统开发中是比较常见的.

多数情况下, 分给平台设备的内存和中断请求号资源是不足以让设备正常工作的. 板设置代码通常会用设备
的platform_data域来存放附加信息, 并向外提供它们.

嵌入式系统时常需要为平台设备提供一个或多个时钟信号. 除非被用到, 这些时钟一般处于静息状态以节电.
系统设置代码也负责为设备提供这些时钟, 以便设备能在它们需要是调用
clk_get(&pdev->dev, clock_name).

老式驱动: 设备枚举
~~~~~~~~~~~~~~~~~~
有些驱动不能被彻底归入驱动模型, 因为它们承担了一个应该由系统基本组建来完成的任务:注册平台设备.
因为热插拔和冷插拔要求由非驱动程序的系统组件来创建设备,所以上述驱动不支持这两种机制.

保留这些驱动的唯一"恰当"的理由是要用它们应付老式的系统设计, 比如原始的IBM PC就依赖一种容易出错
的"枚举硬件(probe-the-hardware)"模型来配置硬件. 新的系统基本上废弃了这种模型, 而倾向对动态配置的总线级支持(PCI, USB), 或是由启动固件来提供设备描述表(例如x86上的PNPACPI). 关于什么东西
出现在哪儿有太多冲突的可能,即使由操作系统来做有根据的猜测, 也难免因频繁出错而惹麻烦.

(译注: Understanding the Linux Kernel一书的第13.1.1.1节, 提到了"枚举硬件"其中的一段或许
可以用来解释最后一句中所谓的"冲突":
```
    尽管访问I/O端口很容易, 检测哪个I/O端口已被分配给I/O设备却并非易事. 对基于ISA总线
    (译注:原始IBM PC采用的总线)的系统来说尤其如此. 通常设备驱动必须盲目地写数据到一些
    I/O端口来探测设备的存在, 然而,如果该端口已经被另外一种设备占用, 这样的操作就可能导
    致系统崩溃...
```

不提倡这种驱动方式. 假如你在升级这样一个驱动, 请尽力把设备列举从驱动中转移到更合适的地方. 这样
做很赚, 因为驱动程序一开始就处在"正常模式", 可以直接使用由即插即用(PNP)设置或平台设备设置代码
所创建的设备..

```
struct platform_device *platform_device_alloc(
                const char *name, int id);
```

你可以用platform_device_alloc()来动态地给设备分配空间, 然后在用platform_device_register()
加上一些资源来初始化该设备.

经常也用另外一个更好的解决办法:

```
struct platform_device *platform_device_register_simple(
                const char *name, int id,
```
~~~~~~~~~~~~~~~~~~

struct resource *res, unsigned int nres);

你可以用platform_device_register_simple()一次完成分配空间和注册设备的任务.

设备命名和驱动绑定
~~~~~~~~~~~~~~~~~
platform_device.dev.bus_id是设备的真名. 它由两部分组成:

*platform_device.name ... 这也被用来匹配驱动

*platform_device.id ... 设备实例号, 或者用"-1"表示只有一个设备.

连接这两项, 像"serial"/0就表示bus_id为"serial.0", "serial"/3表示bus_id为"serial.3";
上面二例都将使用名叫"serial"的平台驱动. 而"my_rtc"/-1的bus_id为"my_rtc"(无实例号), 它的
平台驱动为"my_rtc".

在找到一个设备和驱动的配对后, 驱动绑定是通过调用probe()由驱动核心自动完成的. 如果probe()
成功,
驱动和设备就正常绑定了. 有三种不同的方法来进行配对:

-设备一被注册, 就检查对应总线下的各驱动, 看是否匹配. 平台设备应在系统启动过程的早期被注册

-当驱动通过platform_driver_register()被注册时, 就检查对应总线上所有未绑定的设备.
驱动通常在启动过程的后期被注册或通过装载模块来注册.

-用platform_driver_probe()来注册驱动的效果跟用platform_driver_register()几乎
相同, 不同点仅在于,如果再有设备注册, 驱动就不会再被枚举了. (这无关紧要, 因为这种接口只
用在不可热插拔的设备上.)
~~~~~~~~~~~~~~~~~