Linux内核文档之设备树中文版_鸢飞戾天_新浪博客

Linux and the Device Tree

Linux内核设备树数据使用模型。

Open Firmware Device Tree (DT) 是一个数据结构，也是一种描述硬件的语言。准确地说，它是一种能被操作系统解析的描述硬件的语言，这样操作系统就不需要把硬件平台的细节在代码中写死。

从结构上来说，DT是一个树形结构，或者有名结点组成的非循环图，结点可能包含任意数量的有名属性，有名属性又可以包含任意数量的数据。同样存在一种机制，可以创建从一个结点到正常树形结构之外的链接。

从概念上讲，一套通用的使用方法，即bindings。Bindings定义了数据如何呈现在设备树中，怎样描述典型的硬件特性，包括数据总线，中断线，GPIO连接以及外设等。

尽可能多的硬件被描述从而使得已经存在的bindings最大化地使用源代码，但是由于属性名和结点名是简单字符串， 可以通过定义新结点和属性的方式很方便地扩展已经存在的bindings或者创建一个新的binding。在没有认真了解过已经存在的bindings的情况下，创建一个新的binding要慎之又慎。对于I2C总线，通常有两种不同的，互不相容的bindings出现，就是因为新的binding创建时没有研究I2C设备是如何在当前系统中被枚举的。

1. 历史

略

2. 数据模型

请参考Device Tree Usage章节

2.1 High Level View

必须要认识到的是，DT是一个描述硬件的数据结构。它并没有什么神奇的地方，也不能把所有硬件配置的问题都解决掉。它只是提供了一种语言，将硬件配置从Linux Kernel支持的board and device driver中提取出来。DT使得board和device变成数据驱动的，它们必须基于传递给内核的数据进行初始化，而不是像以前一样采用hard coded的方式。

观念上说，数据驱动平台初始化可以带来较少的代码重复率，使得单个内核映像能够支持很多硬件平台。

Linux使用DT的三个主要原因：

1） 平台识别 (Platform Identification)

2） 实时配置 (Runtime Configuration)

3） 设备植入 (Device Population)

2.2 平台识别

第一且最重要的是，内核使用DT中的数据去识别特定机器。最完美的情况是，内核应该与特定硬件平台无关，因为所有硬件平台的细节都由设备树来描述。然而，硬件平台并不是完美的，所以内核必须在早期初始化阶段识别机器，这样内核才有机会运行特定机器相关的初始化序列。

大多数情况下，机器识别是与设备树无关的，内核通过机器的核心CPU或者SOC来选择初始化代码。以ARM平台为例，setup_arch()会调用setup_machine_fdt()，后者遍历machine_desc链表，选择最匹配设备树数据的machine_desc结构体。它是通过查找设备树根结点的compatible属性并与machine_desc->dt_compat进行比较来决定哪一个machine_desc结构体是最适合的。

Compatible属性包含一个有序的字符串列表，它以确切的机器名开始，紧跟着一个可选的board列表，从最匹配到其他匹配类型。以TI BeagleBoard的compatible属性为例，BeagleBoard xM Board可能描述如下：

compatible = "ti,omap3-beagleboard", "ti,omap3450", "ti,omap3";

compatible = "ti,omap3-beagleboard-xm", "ti,omap3450", "ti,omap3";

在这里，” ti, omap3-beagleboard-xm” 是最匹配的模型，"ti,omap3450"次之，"ti,omap3"再次之。

机敏的读者可能指出，Beagle xM也可以声明匹配"ti,omap3-beagleboard"，但是要注意的是，板级层次上，两个机器之间的变化比较大，很难确定是否兼容。从顶层上来看，宁可小心也不要去声明一个board兼容另外一个。值得注意的情况是，当一个board承载另外一个，例如一个CPU附加在一个board上。（两种CPU支持同一个board的情况）

Compatible属性的另一个值得注意的地方是，它所用到的字符串最好归档在Documnetation/devicetree/bindings中。

在ARM平台上，对于每一个machine_desc，内核会留意任何di_compat列表入口是否出现在compatible属性中，如果有，那么该machine_desc就成为一个候选值。遍历完machine_desc链表之后，setup_machine_fdt()返回最匹配的machine_desc based on which entry in the compatible property each machine_desc matches against. 如果没有匹配的machine_desc被找到，那么返回NULL。

这种方案的原因是，通过观察，大多数情况下，单个machine_desc支持很多用同样SOC或者一个系列SOC的boards。然而，总有一种情况就是某个特定的板子需要特殊的启动代码。特殊情况下，需要在通用的启动代码中明确地甄别某个板子的初始化，这样的操作是非常麻烦且不利于维护的。

反之，compatible列表允许一个通用的machine_desc通过在dt_compat字段添加less compatible值从而支持多个板子。在上述实例中，通用板子可以通过声明兼容"ti,omap3450"和 "ti,omap3"。

PowerPC采用一种稍微不同的方案实现板子的识别，它通过调用每个machine_desc的machine_desc->probe() Hook函数来实现，第一个返回TRUE的machine_desc结构体被返回。然而，这种情况并没有纳入compatible list的优先级别进行考量， 在新的架构中应该予以避免。

2.3 实时配置

大多数情况下，DT是firmware与Kernel之间进行数据通信的唯一方式，所以也用于传递实时的或者配置数据给内核，比如内核参数，initrd映像的地址等。

大多数这种数据被包含在DT的/chosen结点，形如：

```
chosen { bootargs = "console=ttyS0,115200 loglevel=8";
initrd-start = <0xc8000000>;
                initrd-end = <0xc8200000>;
};
```

Bootargs属性包含内核参数，initrd-*属性定义和initrd文件的首地址和大小。Chosen结点可能包含任意数量的描述平台特殊配置的属性。

在早期初始化阶段，体系结构初始化相关的代码会多次调用of_scan_flat_dt() with different helper callbacks去解析设备树数据before paging is setup。Of_scan_flat_dt()遍历设备树，利用helper提取需要的信息。通常，early_init_dt_scan_chosen() helper用于解析Chosen结点，包括内核参数；early_init_dt_scan_root()用于初始化DT地址空间模型；early_init_dt_scan_memory()用于决定可用内存的大小和地址。

在ARM平台，setup_machine_fdt()负责在选取到正确的machine_desc结构体之后进行早期的DT遍历。

2.4 设备植入 (Device Population)

经过板子识别和早期配置数据解析之后，内核进一步进行初始化。期间，unflatten_device_tree()被调用，将DT的数据转换成一种更有效的实时的形式。同时，机器特殊的启动hooks也会被调用，例如machine_desc->init_early()，machine_desc->init_irq()，machine_desc->init_machine()等钩子函数。

通过名称我们可以猜想到，init_early()会在早期初始化时被执行，init_irq()用于初始化中断处理。利用DT并没有实质上改变这些函数的行为和功能。如果DT被提供，那么不管是init_early()还是init_irq()都可以调用任何DT查找函数(of_* in include/linux/of*.h)去获取额外的平台信息。

在DT上下文中最有趣的钩子函数是machine_desc->init_machine()函数，该函数主要负责populating Linux设备模型。在早期内核中，是通过在board support .c 文件中定义一组static clock structures，platform_devices以及其他数据，并全部注册在machine_desc->init_machine()中。使用DT之后，不再是为每个特定平台写死静态设备，而是通过解析DT生成设备链表，并动态地分配设备结构体。

最简单的machine_desc->init_machine()仅仅负责注册platform_devices。一个platform_device被Linux用来定义那些不能被硬件检测的需要做内存映射或者I/O映射的， 以及组合或者虚拟设备等。对于DT而言，没有platform_device这个概念，platform_device可以被简单地认为是DT根结点下的设备结点以及简单内存映射总线结点的子设备结点。

到此，可以做一个DT的简单的lay out。以NVDIA Tegra board的设备树部分内容为例：

```
/{ compatible = "nvidia,harmony", "nvidia,tegra20";
#address-cells = <1>;
#size-cells = <1>;
interrupt-parent = <&intc>;
chosen { };
aliases { };
memory { device_type = "memory";
                reg = <0x00000000 0x40000000>; };
```

```
soc { compatible = "nvidia,tegra20-soc", "simple-bus";
                        #address-cells = <1>;
                        #size-cells = <1>;
                        ranges;
                        intc: interrupt-controller@50041000 { compatible = "nvidia,tegra20-gic";
                                        interrupt-controller;
                                        #interrupt-cells = <1>;
                                        reg = <0x50041000 0x1000>, < 0x50040100 0x0100 >; };
                        serial@70006300 { compatible = "nvidia,tegra20-uart";
                                        reg = <0x70006300 0x100>;
                                        interrupts = <122>; };
                        i2s1: i2s@70002800 { compatible = "nvidia,tegra20-i2s";
                                        reg = <0x70002800 0x100>;
                                        interrupts = <77>;
                                        codec = <&wm8903>; };
                      i2c@7000c000 { compatible = "nvidia,tegra20-i2c";
                                        #address-cells = <1>;
                                        #size-cells = <0>;
                                        reg = <0x7000c000 0x100>;
                                        interrupts = <70>;
                                        wm8903: codec@1a { compatible = "wlf,wm8903";
                                                        reg = <0x1a>;
                                                        interrupts = <347>; }; }; };
sound { compatible = "nvidia,harmony-sound";
                        i2s-controller = <&i2s1>;
                        i2s-codec = <&wm8903>; }; };
```

在machine_desc->machine_init()被调用时，Tegra board会查找DT去决定哪些结点用来创建platform_devices。然而，我们没法通过DT立刻看出一个结点代表何种类型的设备，或者一个结点是否代表了一个设备。

/chosen, /aliases, /memory结点是信息结点，他们不代表设备。

/soc结点的子结点是内存映射设备，但是codec@1a是一个I2C设备，另外，sound结点也不代表一个设备，而是描述了设备如何连接从而构成audio子系统。我之所以知道每个设备是因为我数序硬件设计，但是内核是怎样知道每个结点代表了什么的呢？

这其中的诀窍在于，内核从DT的根结点开始遍历那些有compatible属性的结点。首先，它假设任何包含compatible属性的结点代表了一个设备；其次，它假设根结点下的任何结点是直接挂接在处理器总线上的，或者是一个混杂的没办法描述的系统设备。对于每个这样的结点，Linux都分配并注册一个platform_device结构体，它们反过来会被绑定在一个platform_driver上。

为什么要用platform_device结构体来实例化这些结点呢？这要归因于Linux驱动模型，几乎所有bus_types都认为它的设备是bus controller的子设备。例如，每一个I2C_client都是I2C_master的子设备；每一个spi_device都是SPI总线的子设备。对于USB，PCI，MDIO等总线而言，同样如此。这种层级关系同样出现在DT中，I2C设备结点之出现在I2C总线结点下，SPI，MDIO，USB等同样如此。唯一不需要任何特定类型父设备的设备是platform device，它们快乐地生活在 Linux /sys/devices树下。因此，如果一个DT结点在DT的根结点下，那么最好把它注册成一个platform_device。

Linux板级支持代码调用of_platform_populate(NULL, NULL, NULL)开始从DT根结点发现设备。之所以所有参数都是NULL，是因为它从DT根结点开始遍历，

所以不需要指定起始结点（第一个NULL），parent struct device（最后一个结点），也不使用match table。对于一个只需要注册设备的板子，machine_desc->init_machine()除了调用of_platform_populate()之外，可以不做任何操作。

在Tegra的例子中，platform_devices占据了/soc和/sound结点，但是SoC结点的子结点如何呢？它们也应该被注册成platform_devices吗？通常的做法是在父设备驱动的probe()函数被调用时注册子设备。这样，一个I2C总线设备驱动会为每一个子结点注册一个i2c_client设备结构体， SPI总线驱动会为子结点注册spi_device结构体，对于其他总线而言同样如此。依据该模型，一个绑定Soc结点的驱动会为Soc结点的所有子结点注册platform_device结构体。板级支持代码会分配和注册一个Soc设备，理论上，一个Soc设备驱动会绑定在Soc设备上，并在其probe()钩子函数中为/soc/interrupt-controller, /soc/serial, /soc/i2s, /soc/i2c注册platform_devices结构体。Easy, right?

事实上，注册paltform_devices的子设备为platform_devices是一个通用模型，设备树支持代码反映了该模型，并使上述实例更易于理解。函数of_platform_populate()的第二个参数是一个of_device_id列表，任何匹配该of_device_id列表任何一个entry的结点都可以使得其子结点被注册，以Tegra为例， 可以如此：

```
static void __init harmony_init_machine(void) {

of_platform_populate(NULL, of_default_bus_match_table, NULL, NULL);

}
```

“simple_bus”是定义在ePARP 1.0 Specification中的，作为一个代表简单内存映射总线的属性，这样of_platform_populate()函数可以写成假设simple-bus兼容的结点都可以被转换成设备结构体。然而，我们把它当作一个参数，这样，板级支持代码可以用它来覆盖默认的行为。

Appendix A: AMBA devices

ARM原始核可以被认为是一个挂载在ARM AMBA总线上的设备，它支持硬件检测和电源管理。在Linux中，amba_device结构体和amba_bus_type用来代表原始核设备。然而，值得注意的是，并非所有AMBA总线上的设备都是原始核，对于Linux而言，amba_device和platform_device实例可以被认为是拥有同样bus字段的兄弟姐妹。

当使用DT时，上述情况对of_platform_populate()创建设备造成了困扰，因为它必须决定是把设备注册成platform_device还是amba_device。这给设备创建模型添加了一点复杂性，但是解决方案并不麻烦。如果一个结点兼容”arm, amba-primecell”， 那么of_platform_populate()会把它注册成amba_device，否则注册成platform_device。

分享：