作者

彭东林
pengdonglin137@163.com

平台

TQ2440
Linux-4.10.17

概述

在设备树中我们经常见到诸如"#clock-cells"、"#dma-cells"、"#reset-cells"、"#phy-cells"、"#iommu-cells"、"#pwm-cells"、"#hwlock-cells"、"#io-channel-cells""、"#msi-cells"、"#power-domain-cells"、"#interrupt-cells"、"#cooling-cells"、“#sound-dai-cells”等等，这种属性到底是干啥用的？ 而有些节点会通过下面的属性引用其他设备树节点，如"dmas"、"resets"、"iommus"、"list"、"pwms"、"clocks"、"hwlocks"、"phys"、"gpios"、"cooling-device"、"power-domains"、"interrupts"等，这些属性又是干啥的？

正文

一、我的理解

带有cells字段属性的节点代表了某种资源管理者，以中断控制器节点为例，含有"#interrupt-cells"的节点代表了一种中断资源的管理者，而其他使用"interrupts"属性引用之前的中断控制器节点的设备树节点代表了某种资源的使用者，也就是某个中断资源的使用者，同时在interrupts属性中也描述了所引用中断资源的详细信息，比如中断类型（PPI or SPI）、中断编号、触发类型等等。而#interrupts-cells属性的含义就是要引用该中断管理器的一个中断资源需要的参数个数，比如前面说的“中断类型”、“中断编号”以及“触发类型”就需要三个参数来表示，每个参数之间用空格隔开。

二、demo-controller

设备树既然写的这么清楚，那么Linux内核又是怎么样处理的呢？
为了弄清楚这个问题，我仿照of-dma.c添加了一个非常简单易懂的of-demo.c，我将demo理解为某种资源，然后添加了demo-controller和demo-user两个驱动，分别表示demo资源管理者驱动以及demo资源使用者驱动。此外，需要在设备树中分别为demo资源管理者和demo资源使用者的节点。

对应的驱动我已经上传到github上面了：

```
1.  git@github.com:pengdonglin137/linux-4.10.17.git -b tq2440_dt
```

对应的commit id是f45a337dbde05fb2d23e26b3729111aef090a70e。

1、demo资源管理者节点

```
1.      demo_controller_one: demo_controller@0 {
2.          compatible = "demo_controller,one";
3.          demo-controller;
4.          #demo-cells = <1>;
5.      };
6.
7.      demo_controller_two: demo_controller@1 {
8.          compatible = "demo_controller,two";
9.          demo-controller;
10.         #demo-cells = <2>;
11.     };
12.
13.     demo_controller_three: demo_controller@2 {
14.         compatible = "demo_controller,three";
15.         demo-controller;
16.         #demo-cells = <3>;
17.     };
```

上面表示了三中dem资源管理者，对于每一个demo资源管理者而言，想要引用该管理者的一个demo资源需要的参数个数。
第4行的将#demo-cells赋值为1，表示要获取demo_controller_one这个demo资源管理者的一个demo资源需要1个参数
第10行的2表示获取demo_controller_two这个demo资源管理者的一个demo资源需要2个参数
第16行的3表示获取demo_controller_three这个demo资源管理者的一个demo资源需要3个参数

2、demo资源的使用者

```
    demo_user_one: demo_user@0 {
        compatible = "demo_user,one";
        demos = <&demo_controller_one 1>,
            <&demo_controller_two 2 3>,
            <&demo_controller_two 5 6>;
        demo-names = "one", "two", "three";
    };

    demo_user_two: demo_user@1 {
        compatible = "demo_user,two";
        demos = <&demo_controller_one 2>,
            <&demo_controller_two 3 4>,
            <&demo_controller_three 5 6 7>;
        demo-names = "one", "two", "three";
    };
```

这里列出了两个demo资使用者的节点。
每一个节点的demos属性是对所引用的demo资源详细描述，demo-names是为了增加可读性，对所引用的每一个demo资源进行了命名，这样者驱动中就可以直接使用name来获得对应的demo资源。
注意：这里的demo-names必须跟demos属性中demo资源的顺序相同。

为了尽力做到简单易懂，demo资源管理者对demo资源使用者传递过来的参数只进行非常简单的处理：

- 如果传递1个参数，那么我们直接返回该参数
- 如果传递2个参数，那么我们将这2个参数相乘，将结果返回
- 如果传递3个参数，那么我们将这3个参数相加，然后返回结果

上面的这些运算操作都是在demo资源管理者内进行的。

3、of_demo.c

这个可以认为是core，这个文件提供了供demo controller和demo user调用的接口函数，如demo controller的注册接口以及demo user申请demo资源的接口。

4、demo-controller.c

这个是demo资源管理者的驱动。
在probe函数中会调用of_demo_controller_register注册一个demo controller，同时为其指定了一个xlate函数：of_demo_simple_xlate， 将来对参数的处理就是在该函数中实现的。

5、demo-user.c

这个是demo资源的使用者的驱动。
在probe函数中，会依次处理demos属性中描述的每一个demo资源，然后交给of_demo子

系统处理，of_demo子系统会将其交给对应的demo资源管理者， 最后将返回的结果打印出来。

三、运行

将of-demo.c编译进入内核，将demo-controller.c和demo-user.c编成内核模块。
加载demo-controller.ko：

```
[root@tq2440 mnt]# insmod demo-controller.ko
[   59.262620] demo-controller demo_controller@0: register demo controller s
[   59.265722] demo-controller demo_controller@1: register demo controller s
[   59.273158] demo-controller demo_controller@2: register demo controller s
```

加载demo-user.ko：

```
[root@tq2440 mnt]# insmod demo-user.ko
[   65.123003] args[0]: 1
[   65.123153] demo-user demo_user@0: one: 1
[   65.123808] 2 x 3 = ?
[   65.126449] demo-user demo_user@0: two: 6
[   65.130085] 5 x 6 = ?
[   65.132263] demo-user demo_user@0: three: 30
[   65.137357] args[0]: 2
[   65.138875] demo-user demo_user@1: one: 2
[   65.142820] 3 x 4 = ?
[   65.145432] demo-user demo_user@1: two: 12
[   65.149195] 5 + 6 + 7 = ?
[   65.151697] demo-user demo_user@1: three: 18
```

下一篇我们分析一下背后的实现。