宋牧春： Linux设备树文件结构与解析深度分析(2)

原创：宋牧春 Linux阅码场 2017-08-26

作者简介

宋牧春，linux内核爱好者，喜欢阅读各种开源代码（uboot、linux、ucos、rt-thread等），对于优秀的代码框架及其痴迷。现就职于一家手机研发公司，任职Android BSP开发工程师。

正文开始

前情提要：
宋牧春： Linux设备树文件结构与解析深度分析(1)
征稿和征稿奖励名单：
Linuxer-"Linux开发者自己的媒体"第二月稿件录取和赠书名单
Linuxer-"Linux开发者自己的媒体"首月稿件录取和赠书名单

6. platform_device和device_node绑定

经过以上解析，DeviceTree的数据已经全部解析出具体的struct device_node和struct property结构体，下面需要和具体的device进行绑定。首先讲解platform_device和device_node的绑定过程。在arch/arm/kernel/setup.c文件中，customize_machine()函数负责填充struct platform_device结构体。函数调用过程如图8所示。

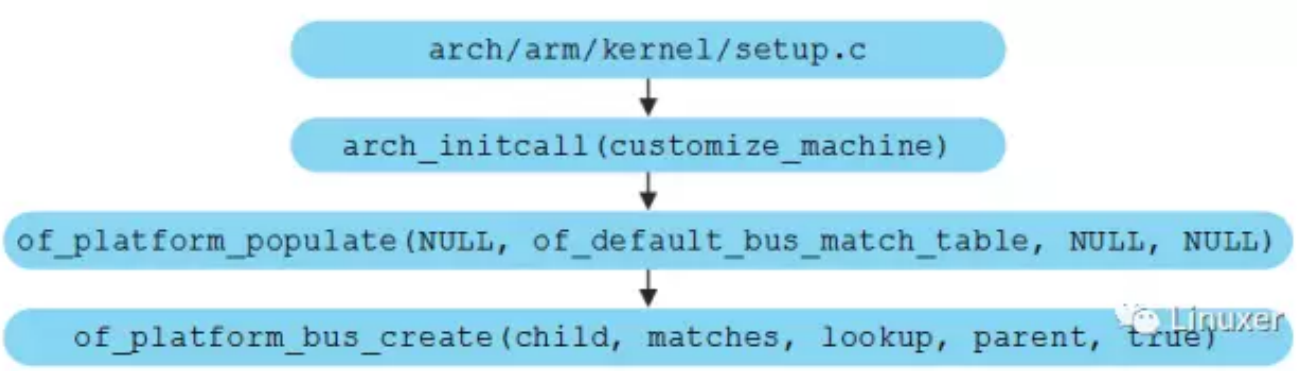

图8 platform_device生成流程图

代码分析如下：

```
const struct of_device_id   of_default_bus_match_table[] = {
       {   .compatible = "simple-bus", },
       {   .compatible = "simple-mfd", },
#ifdef CONFIG_ARM_AMBA
       {   .compatible = "arm,amba-bus", },
#endif /* CONFIG_ARM_AMBA */
       {}   /* Empty terminated list */
};

int of_platform_populate(struct   device_node *root,
                    const   struct of_device_id *matches,
                    const   struct of_dev_auxdata *lookup,
                    struct   device *parent)
{
       struct   device_node *child;
       int   rc = 0;

       /*   获取根节点 */
```

```
	root = root ? of_node_get(root) : of_find_node_by_path("/");
	if (!root)
		return -EINVAL;

	/* 为根节点下面的每一个节点创建platform_device结构体 */
	for_each_child_of_node(root, child) {
		rc = of_platform_bus_create(child, matches, lookup, parent, true);
		if (rc) {
			of_node_put(child);
			break;
		}
	}
	/* 更新device_node flag标志位 */
	of_node_set_flag(root, OF_POPULATED_BUS);

	of_node_put(root);
	return rc;
}

static int of_platform_bus_create(struct device_node *bus,
				  const struct of_device_id *matches,
				  const struct of_dev_auxdata *lookup,
				  struct device *parent, bool strict)
{
	const struct of_dev_auxdata *auxdata;
	struct device_node *child;
	struct platform_device *dev;
	const char *bus_id = NULL;
	void *platform_data = NULL;
	int rc = 0;

	/* 只有包含"compatible"属性的node节点才会生成相应的platform_device结构体 */
	/* Make sure it has a compatible property */
	if (strict && (!of_get_property(bus, "compatible", NULL))) {
		return 0;
	}
	/* 省略部分代码 */
	/*
	 * 针对节点下面得到status = "ok" 或者status = "okay"或者不存在status属性的
	 * 节点分配内存并填充platform_device结构体
	 */
	dev = of_platform_device_create_pdata(bus, bus_id, platform_data, parent);
	if (!dev || !of_match_node(matches, bus))
		return 0;

	/* 递归调用节点解析函数，为子节点继续生成platform_device结构体，前提是父节点
	 * 的“compatible” = “simple-bus”，也就是匹配of_default_bus_match_table结构体中的数据
	 */
	for_each_child_of_node(bus, child) {
```

```
            rc   = of_platform_bus_create(child, matches, lookup, &dev->dev, strict);
            if   (rc) {
                    of_node_put(child);
                    break;
            }
        }
        of_node_set_flag(bus,   OF_POPULATED_BUS);
        return   rc;
}
```

总的来说，当of_platform_populate()函数执行完毕，kernel就为DTB中所有包含compatible属性名的第一级node创建platform_device结构体，并向平台设备总线注册设备信息。如果第一级node的compatible属性值等于“simple-bus”、“simple-mfd”或者"arm,amba-bus"的话，kernel会继续为当前node的第二级包含compatible属性的node创建platform_device结构体，并注册设备。Linux系统下的设备大多都是挂载在平台总线下的，因此在平台总线被注册后，会根据of_root节点的树结构，去寻找该总线的子节点，所有的子节点将被作为设备注册到该总线上。

7. i2c_client和device_node绑定

经过customize_machine()函数的初始化，DTB已经转换成platform_device结构体，这其中就包含i2c adapter设备，不同的SoC需要通过平台设备总线的方式自己实现i2c adapter设备的驱动。例如：i2c_adapter驱动的probe函数中会调用i2c_add_numbered_adapter()注册adapter驱动，函数流执行如图9所示。

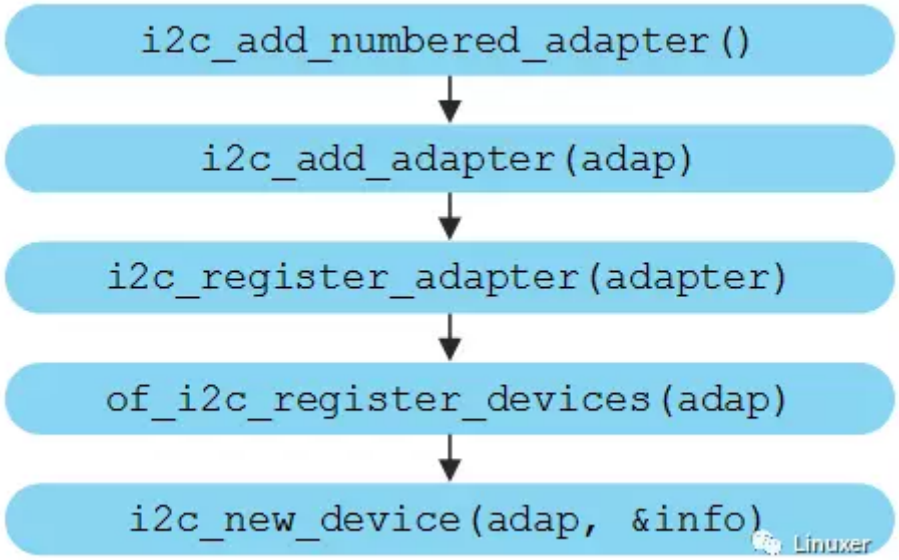

图9 i2c_client绑定流程

在of_i2c_register_devices()函数内部便利i2c节点下面的每一个子节点，并为子节点（status = “disable”的除外）创建i2c_client结构体，并与子节点的device_node挂接。其中i2c_client的填充是在i2c_new_device()中进行的，最后device_register()。在构建i2c_client的时候，会对node下面的compatible属性名称的厂商名字去除作为i2c_client的name。例如：compatible = “maxim,ds1338”,则i2c_client->name = “ds1338”。

8. Device_Tree与sysfs

kernel 启动流程为start_kernel()→rest_init()→kernel_thread():kernel_init()→do_basic_setup()→driver_init()→of_core_init()，在

of_core_init()函数中在sys/firmware/devicetree/base目录下面为设备树展开成sysfs的目录和二进制属性文件，所有的node节点就是一个目录，所有的property属性就是一个二进制属性文件。

精彩直播

《Linux的进程、线程以及调度》4节系列课方案出炉!
《深入探究Linux的设备树》讲座ppt分享和录播地址发布
《Linux总线、设备、驱动模型》直播PPT分享
CSDN Docker实战三小时直播Practical Docker

精彩文章

宋宝华：Linux的任督二脉——进程调度和内存管理
谢宝友: 手把手教你给Linux内核发patch
邢森： 浅析Linux kernel的阅读方法
何晔： 当ZYNQ遇到Linux Userspace I/O（UIO）
笨叔叔： 我的Linux内核学习经历
黄伟亮：ext4文件系统之裸数据的分析实践
徐西宁： 码农小马与Docker不得不说的故事
陈然： 容器生态系统的发展与演变之我见
让天堂的归天堂，让尘土的归尘土——谈Linux的总线、设备、驱动模型
宋宝华：Docker 最初的2小时(Docker从入门到入门)

与其相忘于江湖，不如点击二维码关注Linuxer~