1. 什么是设备树？

设备树是一种描述硬件资源的数据结构。它通过bootloader将硬件资源传给内核，使得内核和硬件资源描述相对独立。

有了初步概念之后，我们再来一起探讨设备树的起源。

/\*

通过这几句话我们知道：设备树是一种描述硬件资源的数据结构。而且

对于设备树来说：内核加载时候，内核里面并没有设备树文件，而是在 内核启动的时候，将设备树文件传递给内核。并没有编译进内核。

而对于平台总线模型：在内核启动的时候就会把平台文件（硬件描述部分）加载进去。在内核编译的时候就加载进入了

也就是说：硬件描述部分与底板相关的代码都被移出内核了，在内核要用到的时候再使用bootloader传递给内核。而使内核不再臃肿。

\*/

1. 设备树的由来？

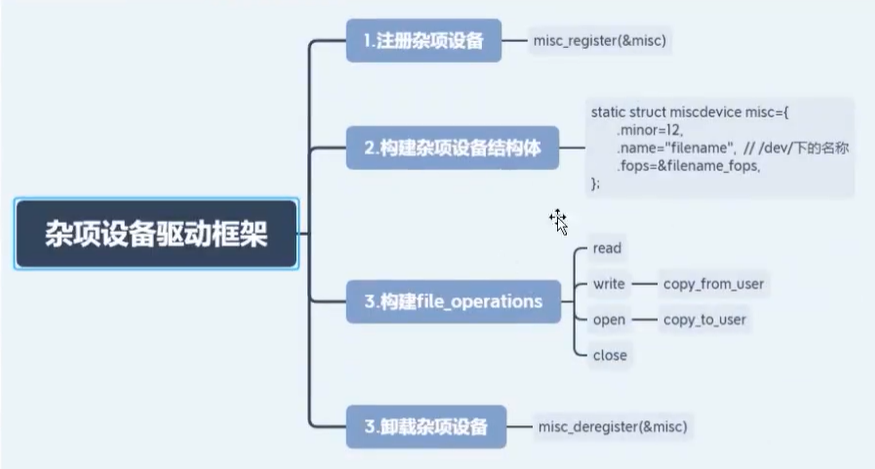
要想了解为什么会有设备树，设备树是怎么来的，我们就要先来回顾一下在没有设备树之前我们是怎么来写一个驱动程序的。以字符设备驱动代码框架为例，我们一起来回顾下。

任何的设备驱动的编写，Linux已经为我们打好了框架，我们只要像做完形填空一样填写进去就可以了。

字符设备驱动框架：



杂项设备驱动框架：



通过这些框架，我们可以很容易编写我们的驱动代码，但是，当我们用这个框架非常熟练的时候，我们就会发现虽然这个方法很简单，但是非常不容易扩展，当我们有很多很多相似设备的时候，如果我们都是按照这个框架来完成，那就要写很多遍这个流程，但是多个相似设备之间真正有差异的地方只有框架的第四步，也就是初始化硬件的部分，其他步骤的代码基本都是一样的。这样就会造成大量的重复代码。但是，我们在编写驱动代码的时候，我们要尽量做到代码的复用，也就是一套驱动尽量可以兼任很多设备，如果我们还按照这个来编写就不太符合我们的规则了。

为了实现这个目标，我们就要把通用的代码和有差异的代码分离出来，来增强我们驱动代码的可移植性。所以，设备驱动分离的思想也就应运而生了，在Linux中，我们是在写代码的时候进行分离，分离是把一些不相似的东西放到了device.c，把相似的东西放在了driver.c，如果我们有很多相似的设备或者平台，我们只要修改dev.c就可以了，这样我们重复性的工作就大大的减少了。这个就是平台总线的由来。

/\*

设备树就是描述设备硬件资源信息的，代替了我们device.c的部分。虽然拿到了内核外面，但匹配方法还是不变的。

\*/

平台总线的方法有什么弊端呢？

当我们用这个方法用习惯以后就会发现，假如soc不变：

我们每换一个平台，都要修改C 文件（开发板地板不一样，device里面对硬件的描述就不一样，那么每换一个地板，就需要重新改写device.c），

并且还要重新编译（编译驱动、内核等）。

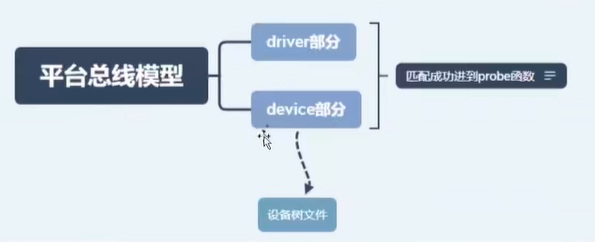
而且会在arch/arm/plat-xxx和arch/arm/mach-xxx下面留下大量的关于板级细节的代码。（会导致内核显得十分臃肿，每换一个平台都需要在arch/arm/plat-xxx 平台总线里面写关于底板的相关硬件描述）

并不是说这个方法不好，只是从Linux的发展来看，这些代码相对于Linux内核来说就是“垃圾代码”，而且这些“垃圾代码”非常多，于是就有了Linux

Torvalds 那句简单粗暴的话:

this whole ARM thing is a fucking pain in the ass

为了改变这个现状，设备树也就被引进到Linux上了，用来剔除相对内核来说的“垃圾代码”，即：用设备树文件来描述这些设备信息，也就是代替dev.c文件，虽然拿到了内核外面，但platform匹配上基本不变，并且相比于之前的方法，使用设备树不仅可以去掉大量“垃圾代码”，并且采用文本格式，方便阅读和修改，如果需要修改部分资源，我们也不用再重新编译内核了，只需要把设备树源文件编译成二进制文件，再通过bootloader传递给内核就可以了。内核在对其进行解析和展开得到一个关于硬件的拓扑图。我们通过内核提供的接口获取设备树的节点和属性就可以了。即内核对于同一soc的不同主板，只需更换设备树文件dtb即可实现不同主板的无差异支持，而无需更换内核文件。



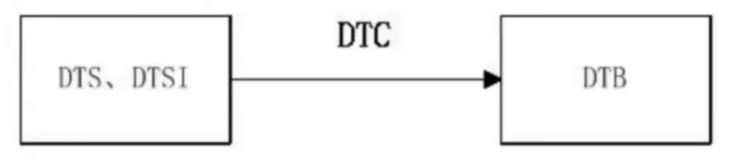
也就是说，把平台总线模型的device.c部分变成了设备树文件。其他的内容都是没有改变的。

1. 设备树的基本概念：
2. 为什么叫设备树？

因为其语法结构像树一样，所以叫它设备树。

1. 常用名词解释:
   1. DT：Device Tree //设备树
   2. FDT：Flattened Device Tree //展开设备树//开放固件，设备树起源于OF，所以我们在设备树中可以看到有很多of字母的函数。
   3. device tree source(dts) //设备树代码
   4. device tree source include(dtsi) //更通用的设备树代码，也就是相同芯片但不同平台都可以使用的代码。
   5. device tree blob(dtb) //DTS编译后得到的DTB文件
   6. device tree compiler(dtc) //设备树编译器

其中，DTS、DTSI、DTB、DTC。他们之间的关系如下：



DTS和DTSI是设备树的源文件 通过DTC编译工具 编译成二进制DTB文件