# Devicetree语法及调试

目录

[Devicetree语法及调试 1](#_Toc536379604)

[1 基本语法: 1](#_Toc536379605)

[2 DTS文件布局(layout): 2](#_Toc536379606)

[3 特殊的、默认的属性: 2](#_Toc536379607)

[3.1 根节点: 2](#_Toc536379608)

[3.2 /memory 2](#_Toc536379609)

[3.3 /chosen 2](#_Toc536379610)

[3.4 /cpus 2](#_Toc536379611)

[3.5 /cpus/cpu\* 2](#_Toc536379612)

[4 引用其他节点: 3](#_Toc536379613)

[5 设备树的调试 3](#_Toc536379614)

[5.1 在根文件系统中查看设备树 3](#_Toc536379615)

[5.2 反汇编生成的dtb文件 3](#_Toc536379616)

## 基本语法:

Devicetree node格式:

[label:] node-name[@unit-address] {

[properties definitions]

[child nodes]

};

Property格式1:

[label:] property-name = value;

Property格式2(没有值):

[label:] property-name;

**Property取值只有3种:**

arrays of cells(1个或多个32位数据, 64位数据使用2个32位数据表示),

string(字符串),

bytestring(1个或多个字节)

示例:

a. Arrays of cells : cell就是一个32位的数据

interrupts = <17 0xc>;

b. 64bit数据使用2个cell来表示:

clock-frequency = <0x00000001 0x00000000>;

c. A null-terminated string (有结束符的字符串):

compatible = "simple-bus";

d. A bytestring(字节序列) :

local-mac-address = [00 00 12 34 56 78]; // 每个byte使用2个16进制数来表示

local-mac-address = [000012345678]; // 每个byte使用2个16进制数来表示

e. 可以是各种值的组合, 用逗号隔开:

compatible = "ns16550", "ns8250";

example = <0xf00f0000 19>, "a strange property format";

## DTS文件布局(layout):

/dts-v1/;

[memory reservations] // 格式为: /memreserve/ <address> <length>; 指定内存中的哪些内容要留给自己使用

/ {

[property definitions]

[child nodes]

};

## 特殊的、默认的属性:

### 根节点:

#address-cells // 在它的子节点的reg属性中, 使用多少个u32整数来描述地址(address)

#size-cells // 在它的子节点的reg属性中, 使用多少个u32整数来描述大小(size)

compatible // 定义一系列的字符串, 用来指定内核中哪个machine\_desc可以支持本设备

// 即这个板子兼容哪些平台

// uImage : smdk2410 smdk2440 mini2440 ==> machine\_desc

model // 咱这个板子是什么

// 比如有2款板子配置基本一致, 它们的compatible是一样的

// 那么就通过model来分辨这2款板子

### /memory

device\_type = "memory";

reg // 用来指定内存的地址、大小

### /chosen

bootargs // 内核command line参数, 跟u-boot中设置的bootargs作用一样，比如可在此设置通过串口输出内核打印信息

### /cpus

/cpus节点下有1个或多个cpu子节点, cpu子节点中用reg属性用来标明自己是哪一个cpu

所以 /cpus 中有以下2个属性:

#address-cells // 在它的子节点的reg属性中, 使用多少个u32整数来描述地址(address)

#size-cells // 在它的子节点的reg属性中, 使用多少个u32整数来描述大小(size)

// 必须设置为0

### /cpus/cpu\*

device\_type = "cpu";

reg // 表明自己是哪一个cpu

## 引用其他节点:

a. phandle : // 节点中的phandle属性, 它的取值必须是唯一的(不要跟其他的phandle值一样)

pic@10000000 {

phandle = <1>;

interrupt-controller;

};

another-device-node {

interrupt-parent = <1>; // 使用phandle值为1来引用上述节点

};

b. label:

PIC: pic@10000000 {

interrupt-controller;

};

another-device-node {

interrupt-parent = <&PIC>; // 使用label来引用上述节点,

// 使用lable时实际上也是使用phandle来引用,

// 在编译dts文件为dtb文件时, 编译器dtc会在dtb中插入phandle属性

};

## 设备树的调试

### 在根文件系统中查看设备树

1. /sys/firmware/fdt // 原始dtb文件

hexdump -C /sys/firmware/fdt

1. /sys/firmware/devicetree // 以目录结构程现的dtb文件, 根节点对应base目录, 每一个节点对应一个目录, 每一个属性对应一个文件
2. /sys/devices/platform // 系统中所有的platform\_device, 有来自设备树的, 也有来有.c文件中注册的

对于来自设备树的platform\_device,可以进入 /sys/devices/platform/<设备名>/of\_node 查看它的设备树属性

1. /proc/device-tree 是链接文件, 指向 /sys/firmware/devicetree/base

### 反汇编生成的dtb文件

**./dtc -I dtb -O dts ../../arch/arm/boot/dts/imx6q-sabresd-ldo.dtb  > ~/f.dts**

使用dtc可对生成的dtb文件进行反汇编，查看编译后对应的最终的dts文件。