Linux 设备树操作API_Linux编程_Linux公社-Linux系统门户网站

Linux 设备树操作API

[日期：2017-02-06]来源：Linux社区　作者：xiaojiang1025

Linux设备树语法详解一文中介绍了设备树的语法，这里主要介绍内核中提供的操作设备树的API，这些API通常都在"include/of.h"中声明。

device_node

内核中用下面的这个结构描述设备树中的一个节点，后面的API都需要一个device_node对象作为参数传入。

```
//include/of.h
 46 struct device_node {
 47         const char *name;
 48         const char *type;
 49         phandle phandle;
 50         const char *full_name;
 51
 52         struct  property *properties;
 53         struct  property *deadprops;    /* removed properties */
 54         struct  device_node *parent;
 55         struct  device_node *child;
 56         struct  device_node *sibling;
 57         struct  device_node *next;      /* next device of same type */
 58         struct  device_node *allnext;   /* next in list of all nodes */
 59         struct  proc_dir_entry *pde;    /* this node's proc directory */
 60         struct  kref kref;
 61         unsigned long _flags;
 62         void    *data;
 63 #if defined(CONFIG_SPARC)
 64         const char *path_component_name;
 65         unsigned int unique_id;
 66         struct of_irq_controller *irq_trans;
 67 #endif
 68 };
```

```
struct device_node
--47-->节点名
--48-->设备类型
--50-->全路径节点名
--54-->父节点指针
--55-->子节点指针
```

查找节点API

```
/**
 * of_find_compatible_node - 通过compatible属性查找指定节点
 * @from - 指向开始路径的节点，如果为NULL，则从根节点开始
 * @type - device_type设备类型，可以为NULL
 * @compat - 指向节点的compatible属性的值（字符串）的首地址
 * 成功：得到节点的首地址；失败：NULL
 */
struct device_node *of_find_compatible_node(struct device_node *from,const char *type, const char *compat);
```

```
/**
 * of_find_matching_node - 通过compatible属性查找指定节点
 * @from - 指向开始路径的节点，如果为NULL，则从根节点开始
 * @matches - 指向设备ID表，注意ID表必须以NULL结束
 * 范例：   const struct of_device_id mydemo_of_match[] = {
                { .compatible = "fs4412,mydemo", },
                {}
```

```
            };
 * 成功：得到节点的首地址；失败：NULL
 */
struct device_node *of_find_matching_node(struct device_node *from,const struct of_device_id *matches);
```

```
/**
 * of_find_node_by_path - 通过路径查找指定节点
 * @path - 带全路径的节点名，也可以是节点的别名
 * 成功：得到节点的首地址；失败：NULL
 */
struct device_node *of_find_node_by_path(const char *path);
```

```
/**
 * of_find_node_by_name - 通过节点名查找指定节点
 * @from - 开始查找节点，如果为NULL，则从根节点开始
 * @name- 节点名
 *  成功：得到节点的首地址；失败：NULL
 */
struct device_node *of_find_node_by_name(struct device_node *from,const char *name);
```

提取通用属性API

```
/**
 * of_find_property - 提取指定属性的值
 * @np - 设备节点指针
 * @name - 属性名称
 * @lenp - 属性值的字节数
 * 成功：属性值的首地址；失败：NULL
 */
struct property *of_find_property(const struct device_node *np, const char *name, int *lenp);
```

```
/**
 * of_property_count_elems_of_size - 得到属性值中数据的数量
 * @np - 设备节点指针
 * @propname  - 属性名称
 * @elem_size - 每个数据的单位（字节数）
 * 成功：属性值的数据个数；失败：负数，绝对值是错误码
 */
int of_property_count_elems_of_size(const struct device_node *np,const char *propname, int elem_size);
```

```
/**
 * of_property_read_u32_index - 得到属性值中指定标号的32位数据值
 * @np - 设备节点指针
 * @propname  - 属性名称
 * @index  - 属性值中指定数据的标号
 * @out_value - 输出参数，得到指定数据的值
 * 成功：0；失败：负数，绝对值是错误码
 */
int of_property_read_u32_index(const struct device_node *np, const char *propname, u32 index, u32 *out_value);
```

```
/**
 * of_property_read_string - 提取字符串（属性值）
 * @np - 设备节点指针
 * @propname  - 属性名称
 * @out_string - 输出参数，指向字符串（属性值）
 * 成功：0；失败：负数，绝对值是错误码
 */
int of_property_read_string(struct device_node *np, const char *propname, const char **out_string);
```

提取addr属性API

```
/**
 * of_n_addr_cells - 提取默认属性“#address-cells”的值
 * @np - 设备节点指针
 * 成功：地址的数量；失败：负数，绝对值是错误码
 */
int of_n_addr_cells(struct device_node *np);
```

```
/**
 * of_n_size_cells - 提取默认属性“#size-cells”的值
 * @np - 设备节点指针
 * 成功：地址长度的数量；失败：负数，绝对值是错误码
 */
int of_n_size_cells(struct device_node *np);
```

```
/**
 * of_get_address - 提取I/O口地址
 * @np - 设备节点指针
 * @index - 地址的标号
 * @size - 输出参数，I/O口地址的长度
 * @flags - 输出参数，类型（IORESOURCE_IO、IORESOURCE_MEM）
 * 成功：I/O口地址的首地址；失败：NULL
 */
__be32 *of_get_address(struct device_node *dev, int index, u64 *size, unsigned int *flags);
```

```
/**
 * of_translate_address - 从设备树中提取I/O口地址转换成物理地址
 * @np - 设备节点指针
 * @in_addr - 设备树提取的I/O地址
 * 成功：物理地址；失败：OF_BAD_ADDR
 */
u64 of_translate_address(struct device_node *dev, const __be32 *in_addr);
```

```
/**
 * of_iomap - 提取I/O口地址并映射成虚拟地址
 * @np - 设备节点指针
 * @index - I/O地址的标号
 * 成功：映射好虚拟地址；失败：NULL
 */
void __iomem *of_iomap(struct device_node *np, int index);
```

```
/**
 * 功能：提取I/O口地址并申请I/O资源及映射成虚拟地址
 * @np - 设备节点指针
 * @index - I/O地址的标号
 * @name - 设备名，申请I/O地址时使用
 * 成功：映射好虚拟地址；失败：NULL
 */
void __iomem *of_io_request_and_map(struct device_node *np, int index, const char *name);
```

提取resource属性API

```
/**
 * of_address_to_resource - 从设备树中提取资源resource（I/O地址）
 * @np - 设备节点指针
 * @index - I/O地址资源的标号
 * @r - 输出参数，指向资源resource（I/O地址）
 * 成功：0；失败：负数，绝对值是错误码
 */
int of_address_to_resource(struct device_node *dev, int index, struct resource *r);
```

提取GPIO属性API

```
/**
 * include/of_gpio.h
 * of_get_named_gpio - 从设备树中提取gpio口
 * @np - 设备节点指针
 * @propname - 属性名
 * @index - gpio口引脚标号
 * 成功：得到GPIO口编号；失败：负数，绝对值是错误码
 */
int of_get_named_gpio(struct device_node *np, const char *propname, int index);
```

提取irq属性API

```
/**
 * of_irq_count从设备树中提取中断的数量
 * @np - 设备节点指针
 * 成功：大于等于0，实际中断数量，0则表示没有中断
 */
int of_irq_count(struct device_node *dev);
```

```
/**
 * of_irq_get - 从设备树中提取中断号
 * @np - 设备节点指针
 * @index - 要提取的中断号的标号
 * 成功：中断号；失败：负数，其绝对值是错误码
int of_irq_get(struct device_node *dev, int index);
```

提取其他属性API

```
/**
 * of_get_mac_address - 从设备树中提取MAC地址
 * @np - 设备节点指针
 * @成功：MAC（6字节）的首地址；失败：NULL
 */
void *of_get_mac_address(struct device_node *np);
```

本文永久更新链接地址：http://www.linuxidc.com/Linux/2017-02/140228.htm