一、ID匹配之框架代码

id匹配（可想象成八字匹配）：一个驱动可以对应多个设备 ------优先级次低

注意事项：

1. device模块中，id的name成员必须与struct platform_device中的name成员内容一致，因此device模块中，struct platform_device中的name成员必须指定
2. driver模块中，struct platform_driver成员driver的name成员必须指定，但与device模块中name可以不相同

```c
/*platform device框架*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/platform_device.h>

//定义资源数组

static void device_release(struct device *dev)
{
    printk("platform: device release\n");
}

struct platform_device_id test_id = {
    .name = "test_device",
};

struct platform_device test_device = {
    .name = "test_device",//必须初始化
    .dev.release = device_release,
    .id_entry = &test_id,
};

static int __init platform_device_init(void)
{
    platform_device_register(&test_device);
    return 0;
}

static void __exit platform_device_exit(void)
{
    platform_device_unregister(&test_device);
}

module_init(platform_device_init);
module_exit(platform_device_exit);
MODULE_LICENSE("Dual BSD/GPL");
```

```c
/*platform driver框架*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/platform_device.h>

static int driver_probe(struct platform_device *dev)
{
    printk("platform: match ok!\n");
    return 0;
}

static int driver_remove(struct platform_device *dev)
{
    printk("platform: driver remove\n");
    return 0;
}

struct platform_device_id testdrv_ids[] =
{
    [0] = {.name = "test_device"},
    [1] = {.name = "abcxyz"},
    [2] = {}, //means ending
};

struct platform_driver test_driver = {
    .probe = driver_probe,
    .remove = driver_remove,
    .driver = {
        .name = "xxxxx", //必须初始化
    },
    .id_table = testdrv_ids,
};

static int __init platform_driver_init(void)
{
    platform_driver_register(&test_driver);
    return 0;
}

static void __exit platform_driver_exit(void)
{
    platform_driver_unregister(&test_driver);
}

module_init(platform_driver_init);
module_exit(platform_driver_exit);
MODULE_LICENSE("Dual BSD/GPL");
```

why

用到结构体数组，一般不指定大小，初始化时最后加{}表示数组结束

设备中增加资源，驱动中访问资源

二、ID匹配之led驱动

三、设备树匹配

设备树匹配：内核启动时根据设备树自动产生的设备 ------ 优先级最高

注意事项：

1. 无需编写device模块，只需编写driver模块
2. 使用compatible属性进行匹配，注意设备树中compatible属性值不要包含空白字符
3. id_table可不设置，但struct platform_driver成员driver的name成员必须设置

XF 说 dt 和dev模块就2种模式
就是这里
匹配上，还像
之前那样获资源
？
（没还获资源得有
相应函数 [illegible]-9）

```c
/*platform driver框架*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/platform_device.h>

static int driver_probe(struct platform_device *dev)
{
    printk("platform: match ok!\n");
    return 0;
}

static int driver_remove(struct platform_device *dev)
{
    printk("platform: driver remove\n");
    return 0;
}

struct platform_device_id testdrv_ids[] =
{
    [0] = {.name = "test_device"},
    [1] = {.name = "abcxyz"},
    [2] = {}, //means ending
};

struct of_device_id test_of_ids[] =
{
    [0] = {.compatible = "xyz,abc"},
    [1] = {.compatible = "qwe,opq"},
    [2] = {},
};

struct platform_driver test_driver = {
    .probe = driver_probe,
    .remove = driver_remove,
    .driver = {
            .name = "xxxxx", //必须初始化
            .of_match_table = test_of_ids,
        },
};

static int __init platform_driver_init(void)
{
    platform_driver_register(&test_driver);
    return 0;
}

static void __exit platform_driver_exit(void)
{
    platform_driver_unregister(&test_driver);
}

module_init(platform_driver_init);
module_exit(platform_driver_exit);
MODULE_LICENSE("Dual BSD/GPL");
```

四、设备树匹配之led驱动

五、一个编写驱动用的宏

```
struct platform_driver xxx = {
    ...
};
module_platform_driver(xxx);
//最终展开后就是如下形式:
static int __init xxx_init(void)
{
        return platform_driver_register(&xxx);
}
module_init(xxx_init);
static void __exit xxx_init(void)
{
        return platform_driver_unregister(&xxx);
}
module_exit(xxx_exit)
```

;