Linux内核编程09期：设备模型和sysfs文件系统

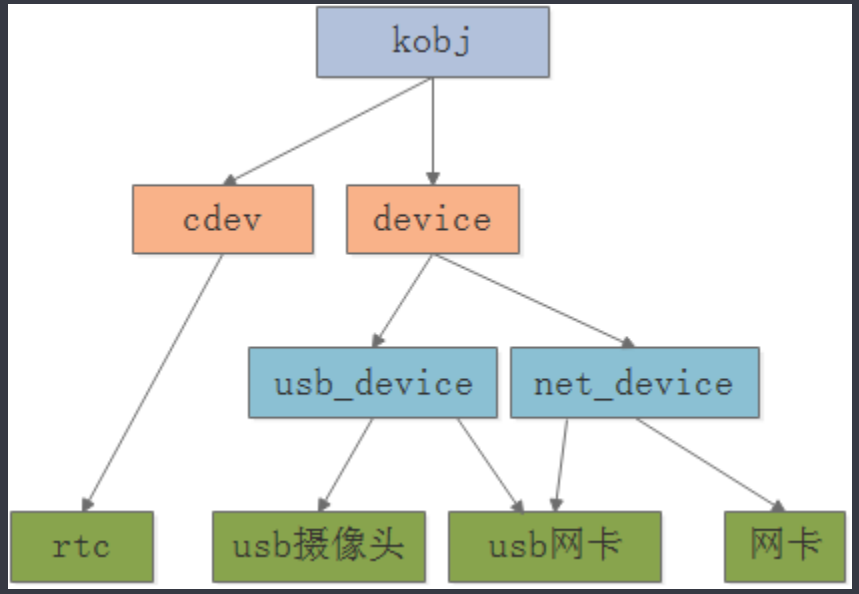
主讲：王利涛

00 什么是设备模型？

主讲：王利涛

- 设备模型简介
  - 什么是设备模型？
  - 设备模型核心数据结构：
    - Kobject、kset、uevent、
    - device、device_driver、bus、class
  - 设备模型的作用
    - 电源管理
    - 热插拔事件管理：hotplug
    - /sys/devices drivers bus
    - 在大山深处盖房子：水电煤气网络
    - 设备模型：在开发区盖厂房

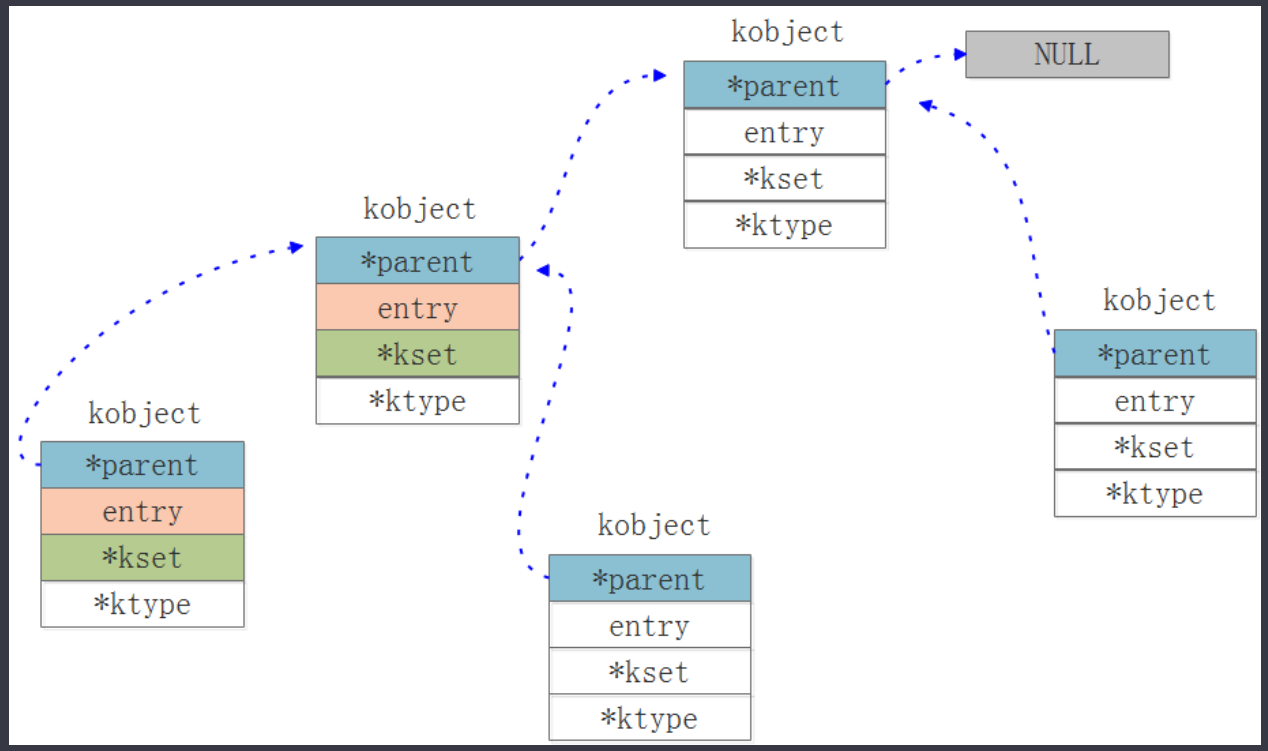
- 设备模型的好处
  - 代码复用：多个设备复用一个驱动
  - 资源的动态申请和释放
  - 简化驱动的编写
  - 热插拔机制：hotplug/uevent
  - 设备模型在内核驱动中的地位
  - 驱动的OOP思想：USB、platform、I2C…



- 本期课程主要内容
  - 设备模型基础：kobject、kset、attribute、uevent
  - 文件系统编程接口：sysfs、注册、挂载、读写
  - 设备模型：device、device_driver、bus、class
  - 设备热插拔事件：hotplug/uevent
  - 如何编写总线型驱动
  - 如何从零实现一个bus子系统：hello bus
  - 如何往bus子系统注册设备、注册驱动
  - 如何实现驱动的复用性：match_table
  - 如何自动创建设备节点
  - 应用程序：mdev/udev 机制分析

01 设备模型基础：kobject

- kobject的主要知识点
  - 结构体定义：struct kobject
  - kobject的作用
  - 对应sysfs下的目录
  - Kobject的初始化和添加流程



• 常用的相关API接口

```
struct kobject *  kobject_create(void);
struct kobject *  kobject_create_and_add (const char *name, struct kobject *parent);

void kobject_init(struct kobject *kobj, struct kobj_type *ktype);
int kobject_add(struct kobject *kobj, struct kobject *parent, const char *fmt, ...);
int kobject_init_and_add(struct kobject *kobj, struct kobj_type *ktype,
                                    struct kobject *parent, const char *fmt, ...);

void kobject_del (struct kobject *kobj);
struct kobject *kobject_get (struct kobject *kobj);
void kobject_put (struct kobject *kobj);
```

02 设备模型基础：attribute（上）

- attribute的主要知识点
  - 关键结构体：attribute、kobj_attribute
  - 初始化宏：__ATTR(_name, _mode, _show, _store)
  - 属性的读写方法：show、store

- 编程示例
  - 在/sys指定目录下创建文件
  - 通过/sys接口修改内核数据
- attribute相关编程接口

```
int  sysfs_create_file (struct kobject *kobj, const struct attribute *attr);
void sysfs_remove_file (struct kobject *kobj, const struct attribute *attr);
```

03 设备模型基础：attribute（下）

- attribute的其他知识点
  - 属性群组：attribute_group
  - 二进制属性：bin_attribute
  - 宏：__ATTRIBUTE_GROUPS(_name)
  - 宏：__BIN_ATTR(_name, _mode, _read, _write, _size)

- 编程示例
  - 在/sys指定目录下创建一组文件
  - 在/sys指定目录下创建二进制文件
- attribute相关编程接口

```
#include <linux/sysfs.h>
int  sysfs_create_bin_file (struct kobject *kobj,  const struct bin_attribute *attr);
void sysfs_remove_bin_file (struct kobject *kobj, const struct bin_attribute *attr);
int  sysfs_create_group (struct kobject *kobj, const struct attribute_group *grp);
void sysfs_remove_group (struct kobject *kobj, const struct attribute_group *grp);
```

04 kobject和sysfs的关联

- 本节主要知识点
  - sysfs文件系统简介
  - sysfs文件系统的注册
  - sysyfs文件系统的挂载
  - kobject和sysfs如何建立关联
    - kobject_create_and_add流程：create_dir
    - sysfs_create_file

05 sysfs目录创建过程分析

- 本节主要知识点
  - kobject和sysfs的关联
  - sysfs目录创建过程：kobject_create_and_add
  - 私有指针的用途

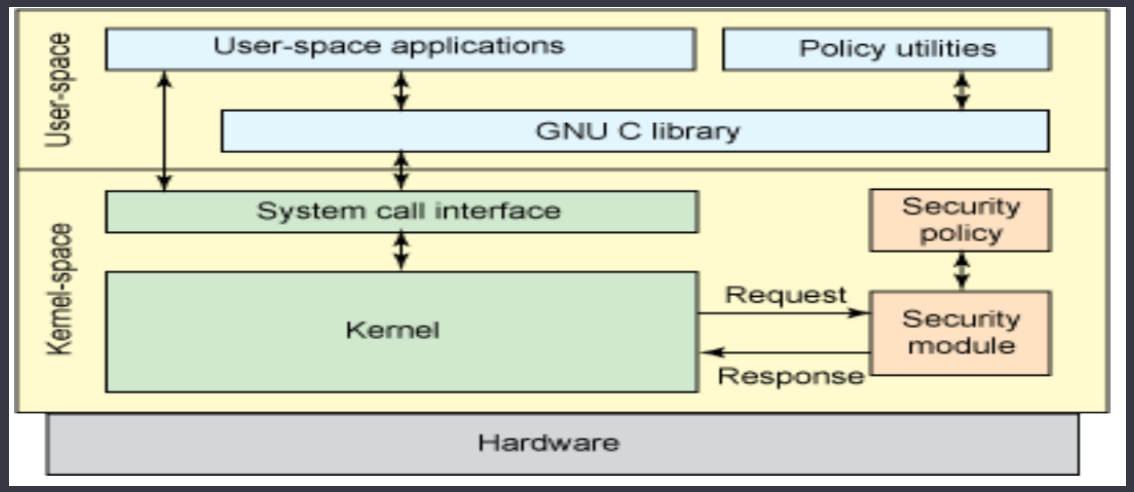
06 sysfs文件创建过程分析

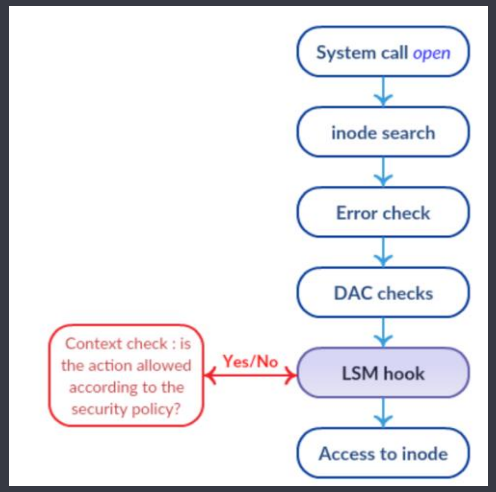
- 本节主要知识点
  - attribute和sysfs的关联
  - sysfs文件创建过程：sysfs_create_file
  - 私有指针的用途
  - 目录和文件创建对比

07 VFS inode的生成过程

- 本节主要知识点
  - VFS层面的inode是如何生成的？
    - 各个文件系统都有自己的inode
    - inode->priv = minix_inode
    - 普通文件：create/touch-syscall-open-inode-minix_inode
  - inode的作用：
    - 包含了一系列操作集：i_fop、i_op
    - 定义了VFS和具体文件系统的各种接口
    - 打开文件时，file->f_op = inode->i_fop
  - inode与kernfs_node的关联

• LSM的hook技术





call_int_hook()
call_void_hook()

08 sysfs文件打开过程分析

- 本节主要知识点
  - 核心结构体之间的关联
  - file指针和inode之间的关联
  - file、seq_file、kernfs_open_file的关联
  - kernfs_open_file和kernfs_inode的关联
  - 私有指针的用途

09 sysfs文件读写过程分析

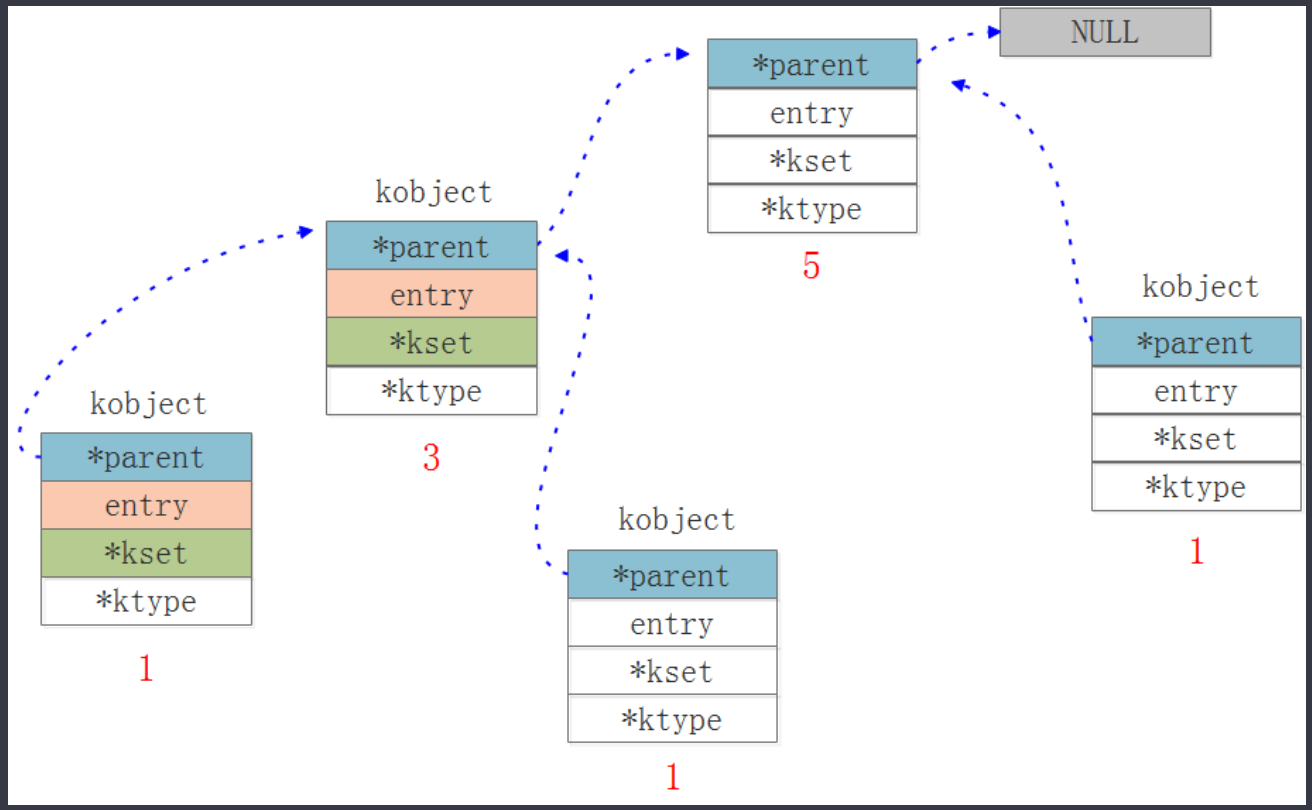
- 本节主要知识点
  - file、seq_file、kernfs_open_file的关联
  - sysfs文件读写过程：
    - 通过cat/echo 修改读写文件
    - show/store回调流程分析

- 思考
  - 为什么再次回调到用户自定义的show/store函数？
    - VFS-minix-minix_read/write-disk
    - VFS-fat-fat_read/write-disk
    - VFS-sysfs-syfs_read/sysfs_write-数据类型各种各样

10 kobject的生命周期

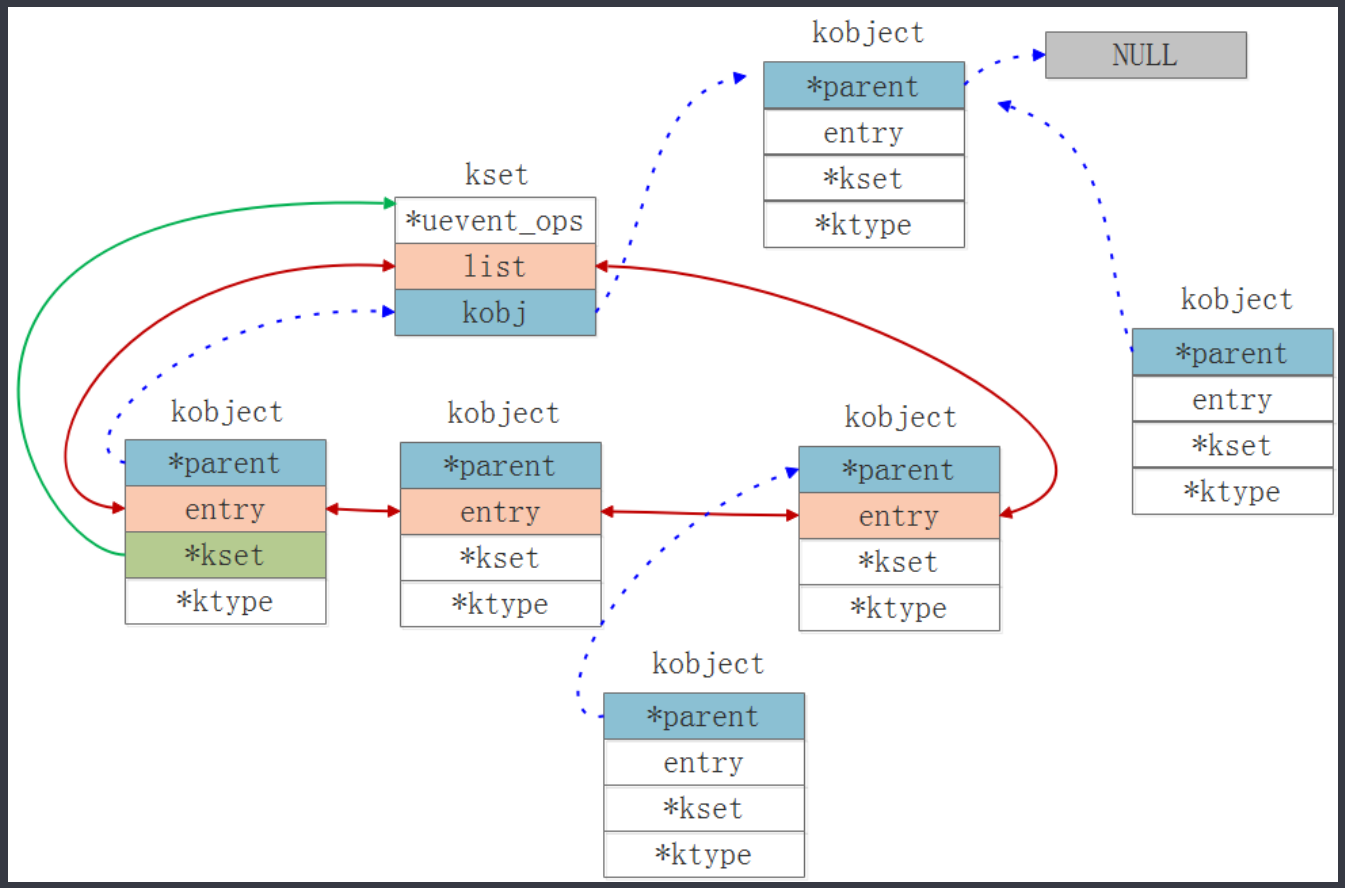
- Kobject的生命周期
  - 为什么要引入生命周期？
  - 内嵌结构体：cdev、device
  - 结构体成员：对应属性
  - 计数接口：
    - void kobject_del (struct kobject *kobj);
    - struct kobject *kobject_get (struct kobject *kobj);
    - void kobject_put (struct kobject *kobj);
    - 回调函数：kobj_type->release

- Kobject引用计数
  - 当添加一个kobject时：自身和parent都加1
  - 当移除一个kobject时：自身和parent都减一
  - 在kobject下创建属性，引用计数不变



11 设备模型基础：kset

- 有关kset的知识点
  - Kset结构体定义
  - Kset的作用
  - Kset与sysfs之间的对应关系



- Kset编程接口及示例
  - 创建一个kset对象
  - 指定kset的parent
  - 创建kset下的属性

```
void kset_init (struct kset *kset);
int    kset_register (struct kset *kset);
void kset_unregister (struct kset *kset);
struct kset *kset_create_and_add (const char *name, const struct kset_uevent_ops *u,
                                  struct kobject *parent_kobj);
```

12 热插拔事件：uevent（上）

- 热插拔(hotplug)的相关知识点
  - 什么是热插拔事件？
  - 什么是uevent？
  - uevent的主要功能是什么？
  - 如何监听uevent？
  - 什么是uevent_helper？
  - udev、mdev是什么？
  - 知识点：
    - » 一个kobject要有kset，然后才能发送uevent
    - » 一个死循环程序，一直监听uevent
    - » 指定要运行程序

- 热插拔（hotplug）uevent编程实验
  - 编写一个内核模块，发送uevent事件
  - 在用户空间监听uevent
  - 内核编译配置：uevent_helper
  - 执行用户空间的指定程序

```
int kobject_uevent (struct kobject *kobj, enum kobject_action action);
enum kobject_action {
        KOBJ_ADD,
        KOBJ_REMOVE,
        KOBJ_CHANGE,
        KOBJ_MOVE,
        KOBJ_ONLINE,
        KOBJ_OFFLINE,
        KOBJ_BIND,
        KOBJ_UNBIND,
};
```

13 热插拔事件：uevent（中）

- 热插拔（hotplug）uevent实现分析
  - 理解kset在发送uevent事件中的作用
    - Kobject是否有kset？
    - Kset中是否定义了uevent？
  - 内核源码分析：uevent热插拔事件处理流程
  - uevent_helper：数组暴露到sysfs用户空间

14 热插拔事件：uevent（下）

- 编程实战：
  - 发送自定义信息到用户空间
  - 编写用户解析工具，解析uevent信息
  - 自动创建设备文件
  - 自动删除设备文件

```
int kobject_uevent_env(struct kobject *kobj, enum kobject_action  action,
                                        char *envp[]);
int add_uevent_var(struct kobj_uevent_env *env, const char *format, ...);
```

15 用OOP思想分析设备模型

- 设备模型中的OOP思想
  - 俄罗斯套娃模式开启：
    - kobject/kset -> device/cdev -> usb_device/net_device
    - kobject_uevent/kobject_add ->xx_register/xx_add
  - OOP思想的C语言模拟实现
    - OOP：封装、继承、多态、抽象类、接口
    - 实现：函数指针、内嵌结构体、私有指针

- 设备模型的核心要素
  - bus： match、uevent、probe、suspend
  - device： kobject、parent、devt、init_name
  - device_driver： name、probe、match_table
  - class： name
  - 设备模型的工作机制：
    - 注册一个总线：实现match方法
    - 往总线注册设备
    - 往总线注册驱动
    - 驱动的probe

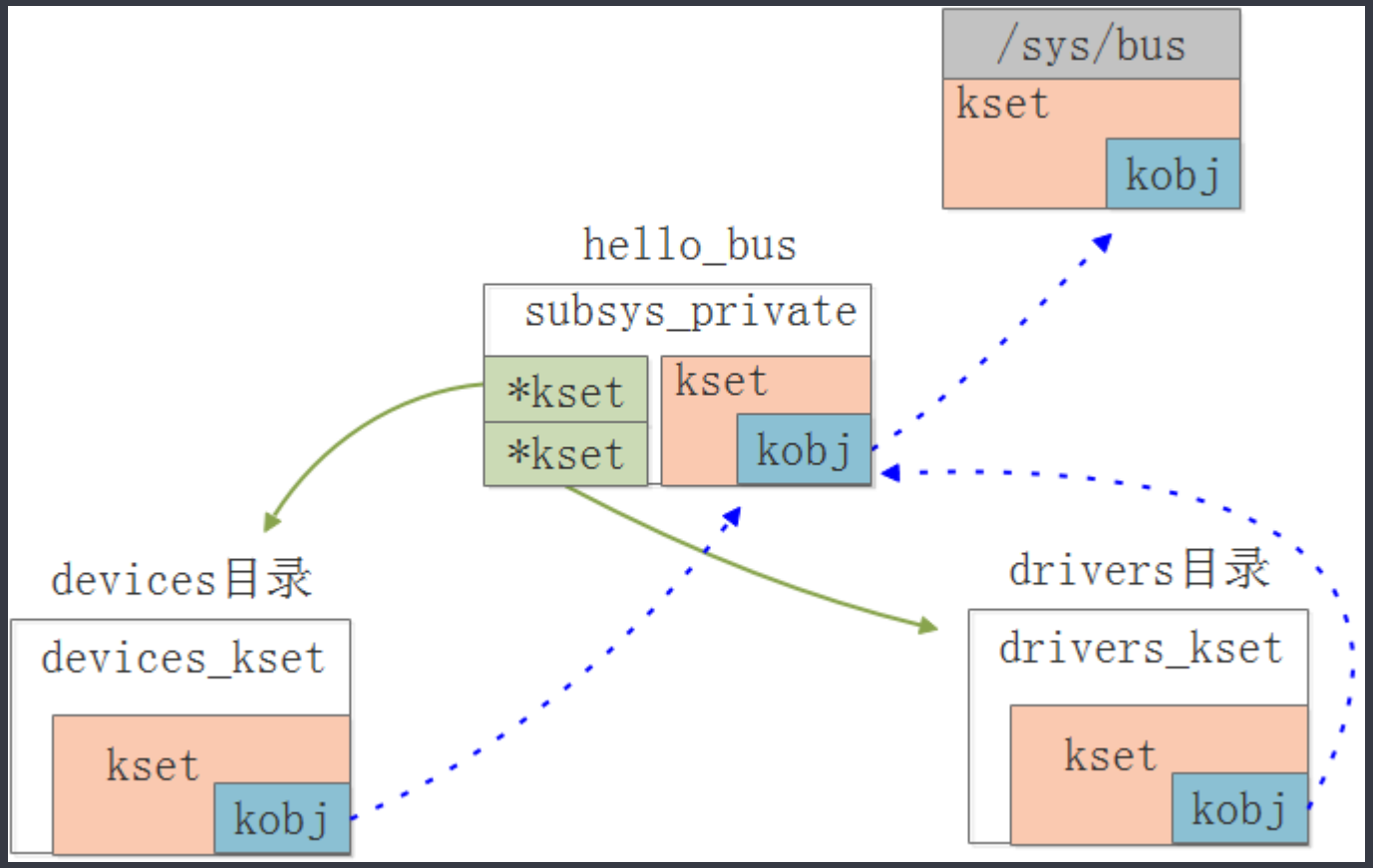
16 设备模型：bus（上）

- 本节主要知识点
  - 结构体：
    - struct bus_type
    - struct bus_attribte
  - 如何注册一个总线
    - 注册接口：bus_register
    - 实现总线的match方法
    - 在bus下创建属性文件

17 设备模型：bus（下）

- 本节主要知识点
  - bus和kobject/kset的关联
    - 在sysfs下的一一对应关系
    - bus：内嵌kset
  - 函数：bus_register 注册流程分析
  - 函数：bus_create_file 过程分析
    - uevent
    - drivers_autoprobe
    - drivers_probe
    - hello_bus_attr

• 内核中的数据结构关联



18 设备模型：device（上）

- 本节主要知识点
  - 结构体：device
  - 结构体：device_attribute
  - 如何向bus上注册设备：device_register
  - 如何创建设备属性文件

19 设备模型：device（下）

- 本节主要知识点
  - 设备注册过程分析： device_register
  - 创建设备属性文件： device_create_file
    - 默认属性文件：uevent、dev
    - 默认链接文件：subsystem
    - 自定义属性文件：hello_device_attr
  - 设备device与总线的关联
  - 设备device与sysfs的关联

20 设备模型：device_driver（上）

- 本节主要知识点
  - 结构体：device_driver
  - 结构体：driver_attribute
  - 如何向总线添加一个驱动：driver_register
  - 如何创建驱动属性文件：driver_create_file

21 设备模型：device_driver（下）

- 本节主要知识点
  - 驱动注册过程分析： driver_register
  - 驱动的auto_probe过程
  - 驱动属性文件创建分析： driver_create_file
  - 核心结构体之间的关联
    - device_driver与bus_type
    - device_driver与kobject、kset

22 bus probe和driver probe

- probe机制
  - probe和init机制对比
  - bus的probe
  - driver的probe

23 设备模型：class（上）

- 关于class的知识点
  - 设备模型中为什么需要class？
  - 有什么作用？
    - 对某类设备的一个抽象，封装出一些标准的接口
    - 字符驱动为例：RTC，file_operations
    - 系统调用：ioctl，cmd，set_time、set_alarm
    - 抽象出一个rtc_class，设置时间、设置闹钟
  - 编程接口：class_create、class_create_file
  - 编程示例：
    - 如何去创建类
    - 如何去创建设备

24 设备模型：class（下）

- 本节学习重点
  - 驱动案例分析：class在内核驱动中的应用
  - class 接口的抽象：rtc_class_ops
    - set_time
    - set_alarm
  - rtc_device的抽象
    - cdev：注册一个字符设备驱动
    - device：通过bus调用对应的probe
    - rtc_class_ops：通过该接口，回调具体RTC的set_time函数

25 device的二次抽象

- 本节主要知识点
  - 内核中的OOP思想
  - 如何对device进一步抽象：hello_device
  - 如何对device_driver进一步抽象：hello_driver
  - 设备注册接口的进一步封装：hello_deivce_register
  - 驱动注册接口的进一步封装：hello_driver_register

26 实现一个总线子系统

- 本节主要知识点
  - 如何实现一个子系统：hello bus
  - 接口的封装
    - 往总线注册设备
    - 往总线注册驱动
    - 总线功能实现：match、probe…
  - 头文件声明：接口进行声明
  - 编程示例

27 驱动复用： match_table

- 本节主要知识点
  - 进一步完善子系统，增加功能
  - 驱动复用：多个设备共享同一个驱动

28 设备的热插拔(hotplug)机制

- 设备的热插拔机制
  - device 热插拔实现机制：uevent
  - 自动创建设备节点
  - 应用程序：udev
  - 嵌入式应用程序：mdev -d
    - uevent_helper：/sbin/mdev

29 从字符驱动到总线驱动（上）

- 将RTC字符驱动升级为总线型驱动
  - 从字符型驱动升级为device_driver 总线型驱动
  - 支持用户接口：设置时间、闹钟
  - 中断线程化
  - 理解初始化流程、probe流程
  - 将RTC注册到hello bus 子系统
    - 将RTC device注册到hello bus
    - 将RTC driver注册到hello bus
  - 自动创建设备节点

30 从字符驱动到总线驱动（下）

- 使用uevent机制实时更新时间
  - 驱动中发送uevent到用户空间
  - 用户应用程序监听uevent
  - 解析uevent信息，自动创建设备节点
  - 实时显示RTC系统时间

31 本期课程小结

- 本期课程小结
  - 实现了一个简单的hello bus子系统
  - 内核中的OOP编程思想
  - 总线型驱动的编写方法
  - 核心基础：platform、USB、I2C…