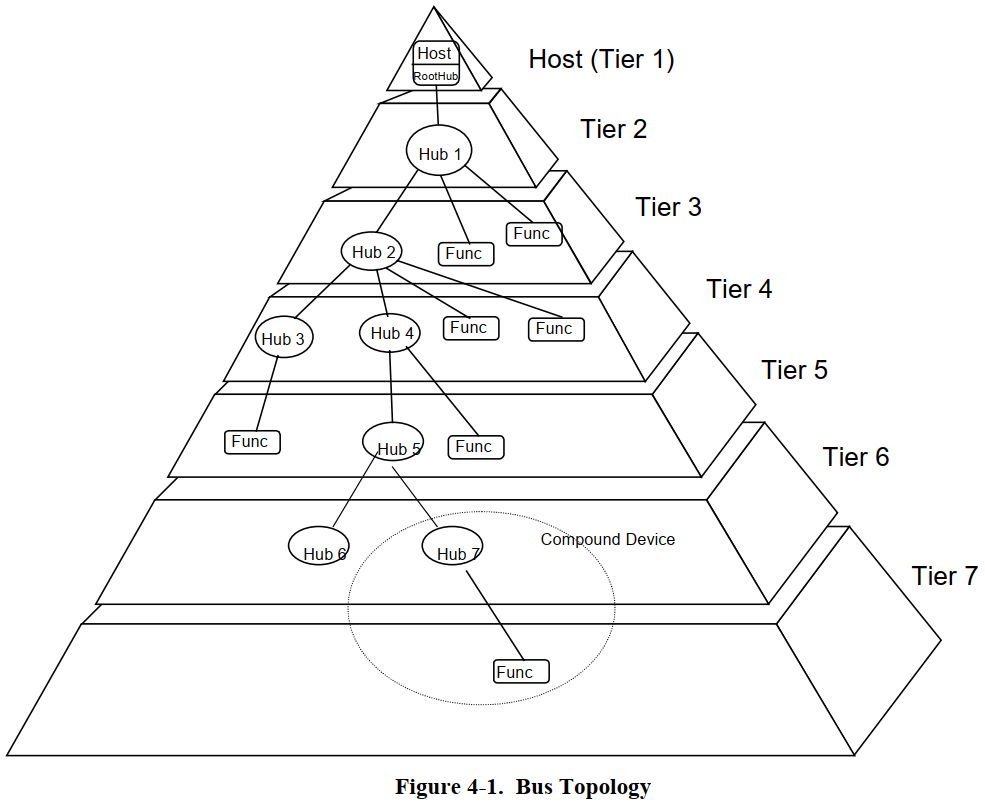
[Linux USB 3.0驱动分析（四）—— USB Core分析](https://www.cnblogs.com/wen123456/p/14281890.html)

**一.拓扑结构**



·之所以要规定这个树形拓扑结构是为了避免环形连接。

·一条USB总线有且只有一个USBHost，对应一个RootHub

·USB设备分为两类，Hub和Functions，Hub通过端口Port连接更多USB设备，Functions即USB外接从设备。

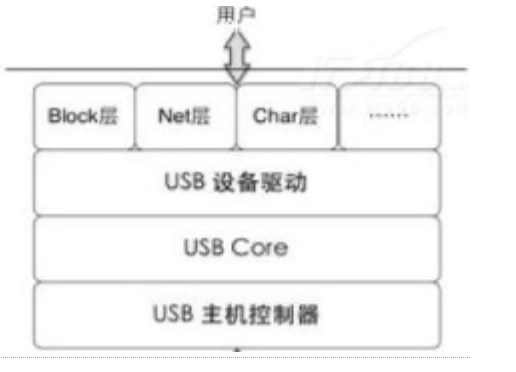
·层次最多7层，且第7层不能有Hub，只能有functions。

·CompoundDevice-一个Hub上接多个设备组成一个小设备。

·CompositeDevice-一个USB外接设备具有多个复用功能。

**二.USBCore**

**1.USB子系统结构**

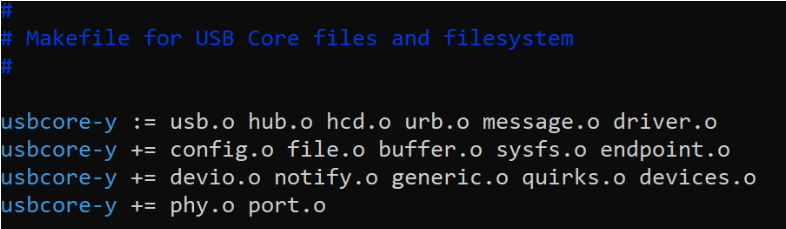


**HCD(Host Controller Device): USB主控制器设备**

协议里说，HCD提供主控制器驱动的硬件抽象，它只对USBCore一个负责，USBCore将用户的请求映射到相关的HCD，用户不能直接访问HCD。换句话说，USBCore就是HCD与USB设备唯一的桥梁。

**2.USB子系统的初始化**

USBcore源码位于./drivers/usb/core，其中的Makefile摘要如下，



usbcore这个模块代表的不是某一个设备，而是所有USB设备赖以生存的模块，它就是USB子系统。

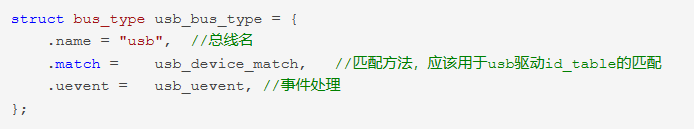


usbcore注册了USB总线，USB文件系统，USBHub以及USB的设备驱动usb generic driver等。

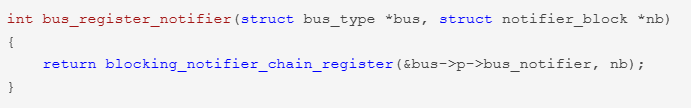
**3. usb\_debugfs\_init --usb debugfs初始化**



**4.usb总线的注册**



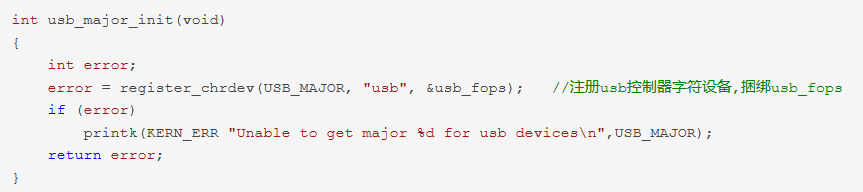
**5.注册usb总线通知链**



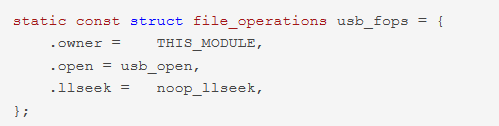
当总线添加删除设备的时候会调用usb\_bus\_nb指定的notifier\_cal方法,既usb\_bus\_notify



**6.初始化usb主控器字符设备,USB\_MAJOR=180**

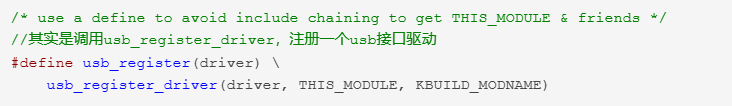


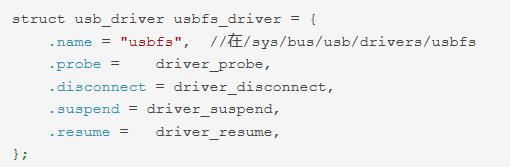
捆绑usb\_fops



**7.注册usbfs\_driver**

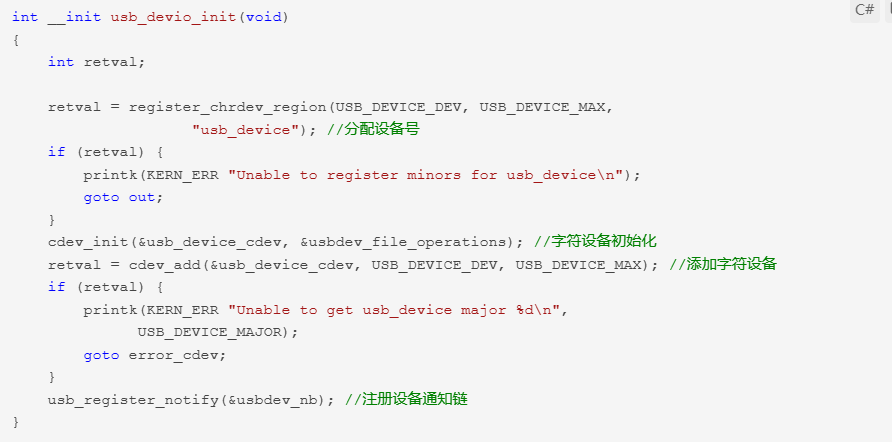
**usb\_register(&usbfs\_driver); //通过usb\_register()将usbfs驱动提交个设备模型，添加到USB总线的驱动链表里。**



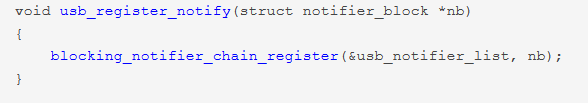


**8.usb\_devio\_init     USB\_DEVICE\_DEV=189**

**//用于应用程序直接访问usb设备**



usb注册的通知链

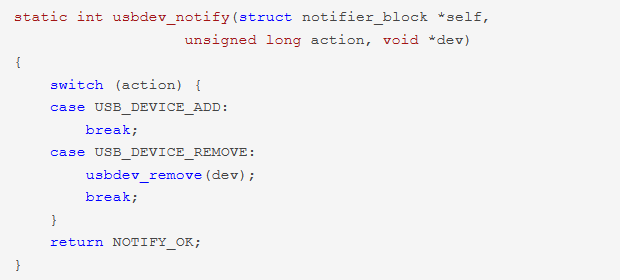


usb通知链表头为usb\_notifier\_list

在/drivers/usb/core/notify.c文件中,有四个函数()对usb\_notifier\_list中发送通知,这四个函数如下:

usb\_notify\_add\_device    //有设备添加  
usb\_notify\_remove\_device    //有设备移除  
usb\_notify\_add\_bus    //总线添加  
usb\_notify\_remove\_bus    //总线移除

当这些事件发生后会调用usbdev\_nb指定的notifier\_cal方法,既usbdev\_notify



**8.usb hub初始化**

**后续详细分析**

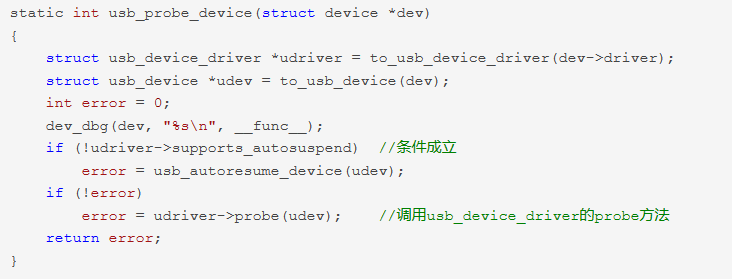
**9. usb注册通用设备驱动**



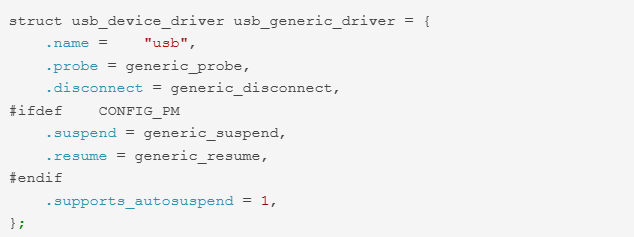
usb\_device\_driver结构体是usb\_driver的简化版本,这里注册的是usb设备(非接口)驱动。

usb总线的match方法对usb设备和usb接口做了区分处理,针对usb设备,直接match的,(分析match时候再细化)

然后调用usb设备驱动的probe方法



接着调用usb\_generic\_driver的probe方法



**对应到root hub,流程会转入到generic\_probe().代码如下:**



**三.USB总线**

**注册USB总线通过bus\_register(&usb\_bus\_type);**

struct bus\_type usb\_bus\_type={

    .name="usb",

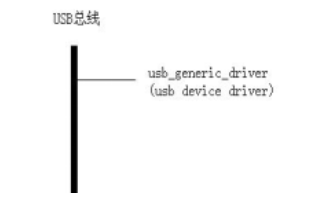
    .match=usb\_device\_match,//这是个很重要的函数，用来匹配USB设备和驱动。

    .uevent=usb\_uevent,.pm=&usb\_bus\_pm\_ops,

};

下面总结下USB设备和驱动匹配的全过程，

**->step1-usb device driver**

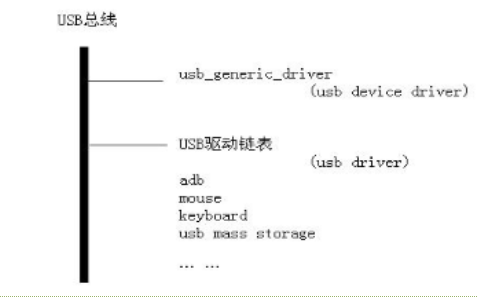


USB子系统初始化的时候就会注册usb\_generic\_driver,它的结构体类型是usb\_device\_driver，它是USB世界里唯一的一个USB设备驱动，区别于struct usb\_driver USB驱动。

·USB设备驱动(usb device driver)就只有一个，即usb\_generice\_driver这个对象，所有USB设备都要绑定到usb\_generic\_driver上，它的使命可以概括为：为USB设备选择一个合适的配置，让设备进入configured状态。

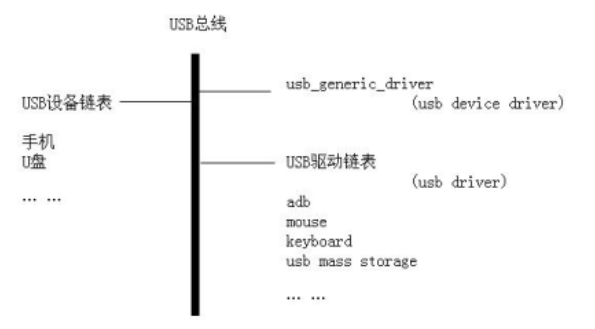
·USB驱动(usb driver)就是USB设备的接口驱动程序，比如adb驱动程序，u盘驱动程序，鼠标驱动程序等等。

**->step2-usb driver**



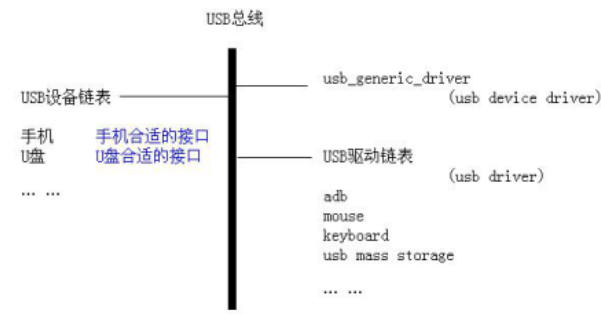
Linux启动时注册USB驱动，在xxx\_init()里通过usb\_register()将USB驱动提交个设备模型，添加到USB总线的驱动链表里。

**->step3-usb device**



USB设备连接在Hub上，Hub检测到有设备连接进来，为设备分配一个struct usb\_device结构体对象，并将设备添加到USB总线的设备列表里。

**->step4-usb interface**



USB设备各个配置的详细信息在USB core里的漫漫旅途中已经被获取并存放在相关的几个成员里。

usb\_generic\_driver得到了USB设备的详细信息，然后把准备好的接口送给设备模型，Linux设备模型将接口添加到设备链表里，然后去轮询USB总线另外一条驱动链表，针对每个找到的驱动去调用USB总线的match函数，完成匹配。