[Linux USB 3.0驱动分析（五）——USB Hub代码分析](https://www.cnblogs.com/wen123456/p/14281906.html)

**本文分析的是linux-5.4.3**

**一、Linux 下USB Hub热插拔处理**

**1、 Linux下USB HUB的驱动的实现和分析：**

       在系统初始化的时候在**usb\_init函数**中调用**usb\_hub\_init函数**，就进入了**hub的初始化**。

**代码路径：drivers\usb\core\hub.c**

**在usb\_hub\_init函数中完成了注册hub驱动**，并且利用**函数alloc\_workqueue创建一个工作队列**。

       USB设备是热插拔，这就和PCI设备不同，PCI设备是在系统启动的时候都固定了，因此PCI设备只需要初始化进行枚举就可以了，采用递归算法即可。而USB设备需要热插拔，因此在**hub\_probe函数**中调用**hub\_configure函数**来配置hub，在这个函数中主要是利用函数**usb\_alloc\_urb函数**来分配一个urb，利用**usb\_fill\_int\_urb**来初始化这个urb结构，包括hub的中断服务程序hub\_irq的，查询的周期等。

       每当有设备连接到USB接口时，USB总线在查询hub状态信息的时候会触发hub的中断服务程序hub\_irq, 在该函数中置位event\_bits，运行工作队列。进入hub\_event函数，该函数用来处理端口变化的事件。然后通过一个for循环来检测每个端口的状态信息。利用**usb\_port\_status**获取端口信息，如果发生变化就调用**hub\_port\_connect\_change**函数来配置端口等。

**2、软件层次分析-初始化**

这里我们先讲讲USB热插拔事件的处理工作。hub\_event工作，运行于工作队列。

hub\_event来检查usb port的事件通知HCD和usb core，然后做相应的处理。

驱动目录**drivers/usb/\***  
usb/serial  usb 串行设备驱动 （例如usb 3G卡、蓝牙等）  
usb/storage  usb 大储量磁盘驱动（u盘）    
usb/host usb host usb主机控制器驱动（嵌入式otg：dwc\_otg）  
usb/core   usb 核心一些处理代码，所有的驱动相关处理都在这里，也都注册到它里面。  
usb/usb-skeleton.c 经典的usb客户驱动框架，可以参考

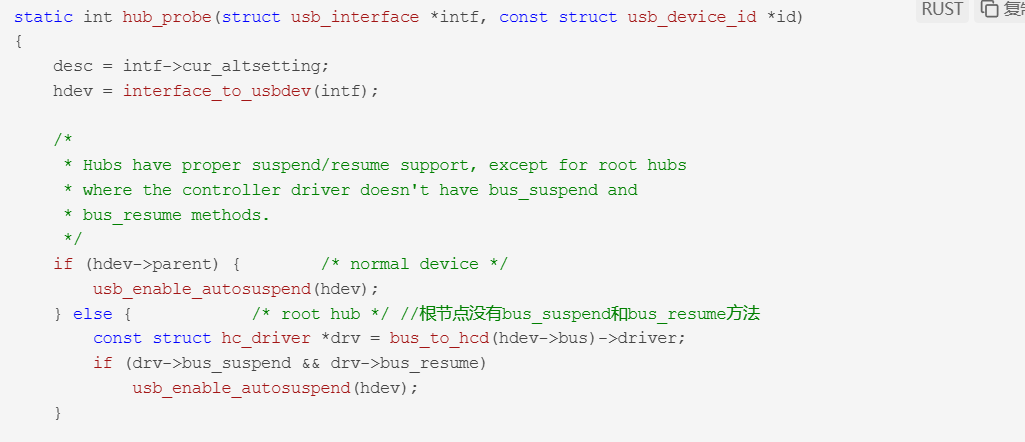
当然还有其他这里不再说明。

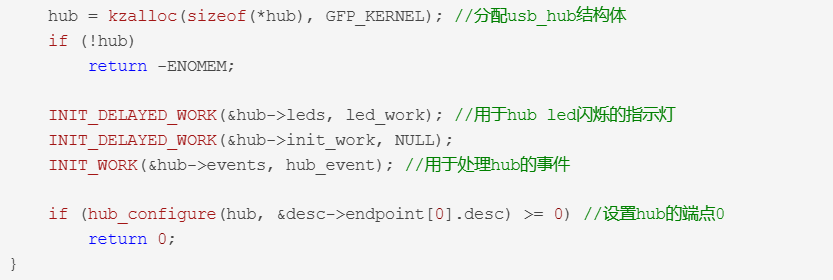
这里我们主要分析khub的工作原理： 硬件层次是hub的工作，如何和host及其设备间通信及相应事件

**[usb/core/hub.c ]**



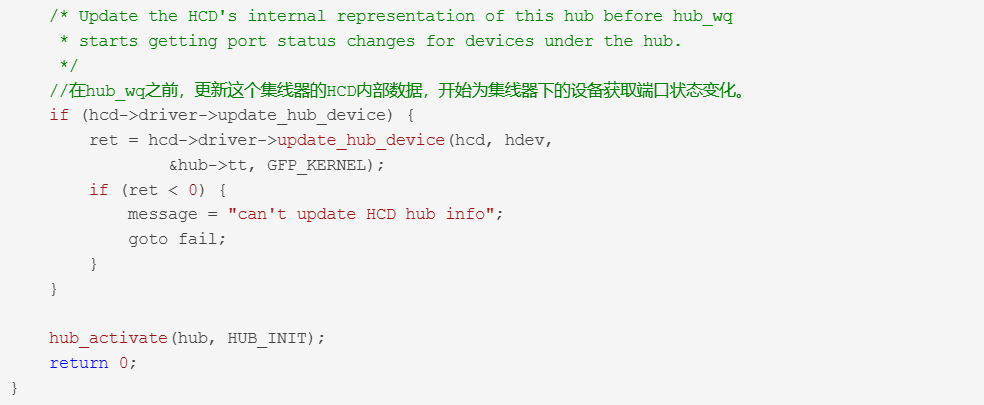
然后进入hub的probe函数，主要是一些工作的初始化和hub的配置，我这里化简了



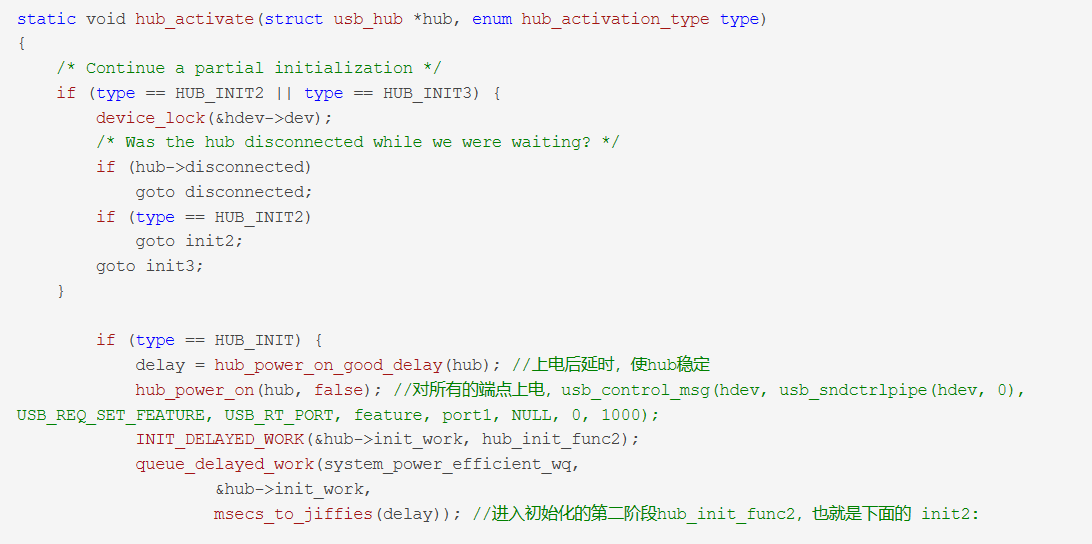


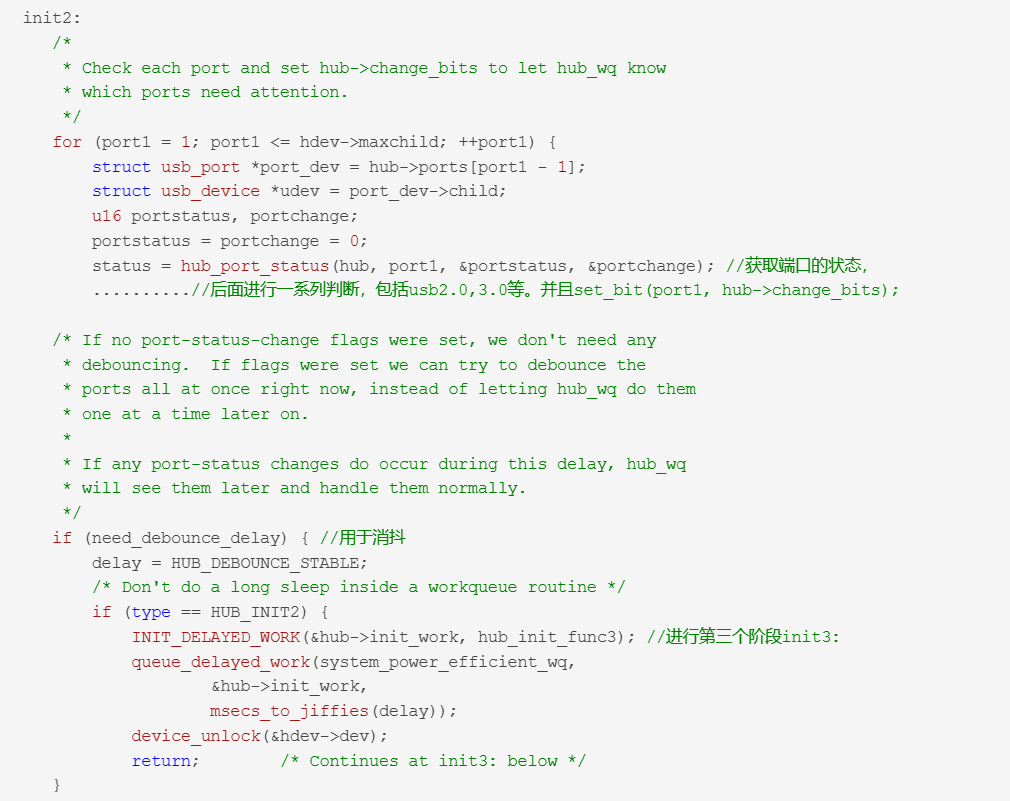
hub\_configure配置hub，包括不同的hub的判断和配置，但Linux认为最多只能31个接口，里面比较复杂，这里也化简了

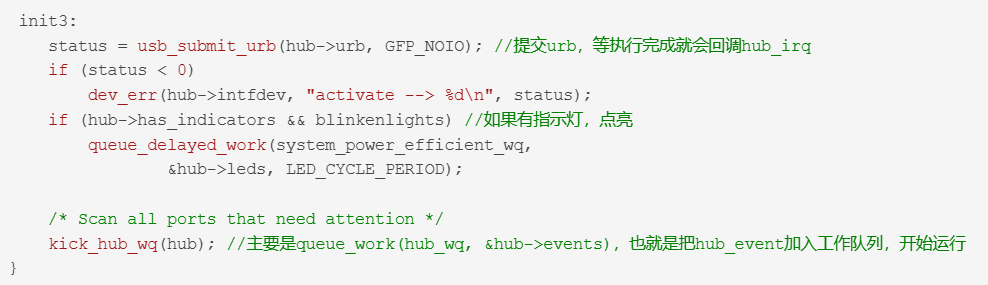




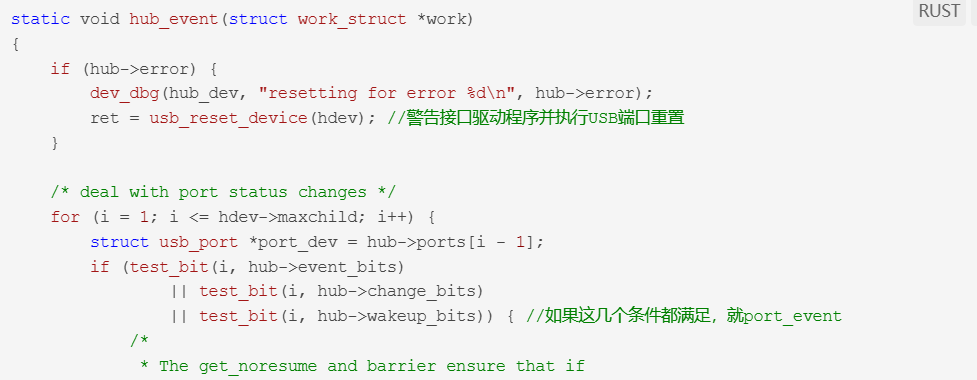
继续分析hub\_activate，主要是启动hub，我们这里传入的参数是HUB\_INIT

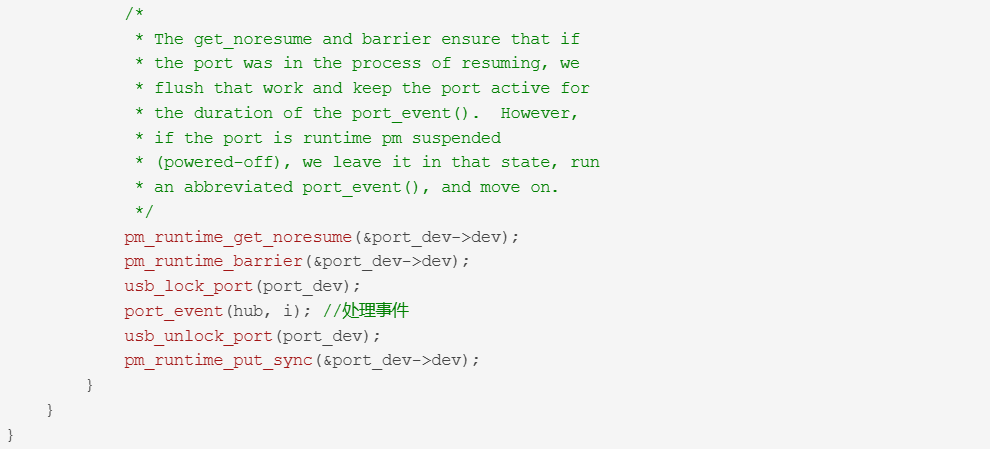






我们来看看hub\_event，前面int2的时候有设置hub->change\_bits，这里会进行处理





我们再看看 port\_event做了什么。

来看一下,什么情况下, hub\_port\_connect\_change才会被设为1.

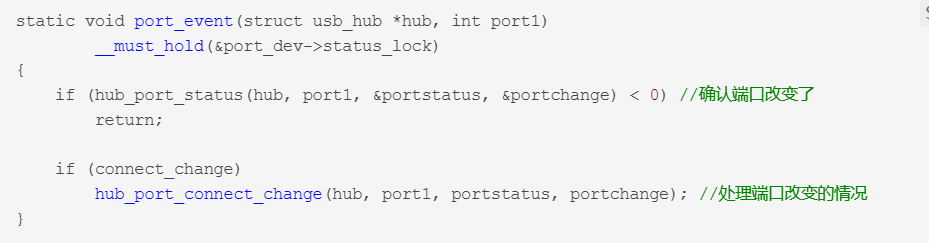
1:端口在hub->change\_bits中被置位.搜索整个代码树,发生在设置hub->change\_bits的地方,只有在hub\_port\_logical\_disconnect()中手动将端口禁用,会将对应位置1.

2:hub上没有这个设备树上没有这个端口上的设备.但显示端口已经连上了设备

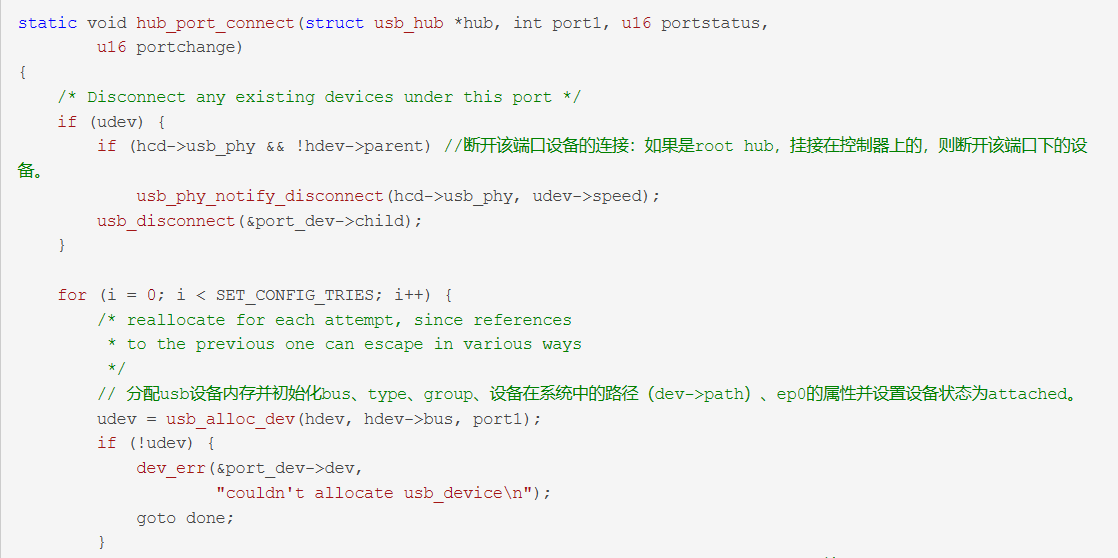
3:hub这个端口上的连接发生了改变,从端口有设备连接变为无设备连接,或者从无设备连接变为有设备连接.

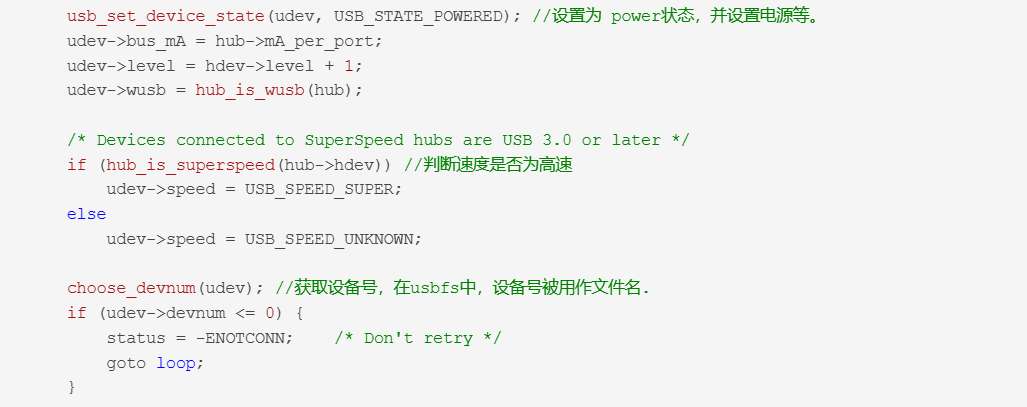
4:hub的端口变为了disable,此时这个端口上连接了设备,但被显示该端口已经变禁用,需要将connect\_change设为1.

5:端口状态从SUSPEND变成了RESUME,远程唤醒端口上的设备失败,就需要将connect\_change设为1.



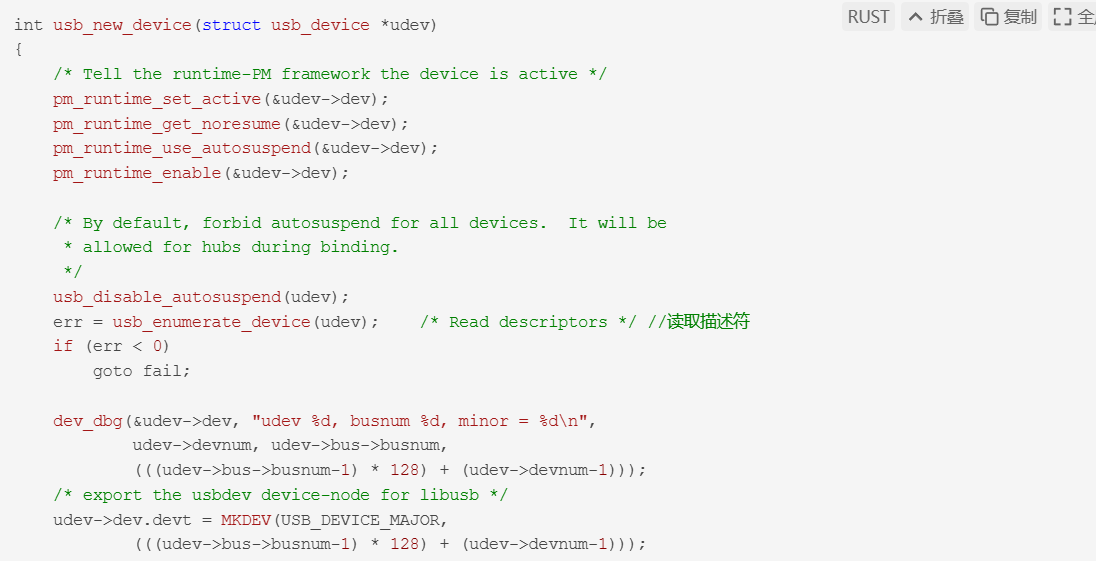
调用hub\_port\_connect\_change再调用hub\_port\_connect







我们重点分析一下usb\_new\_device，初始化设备







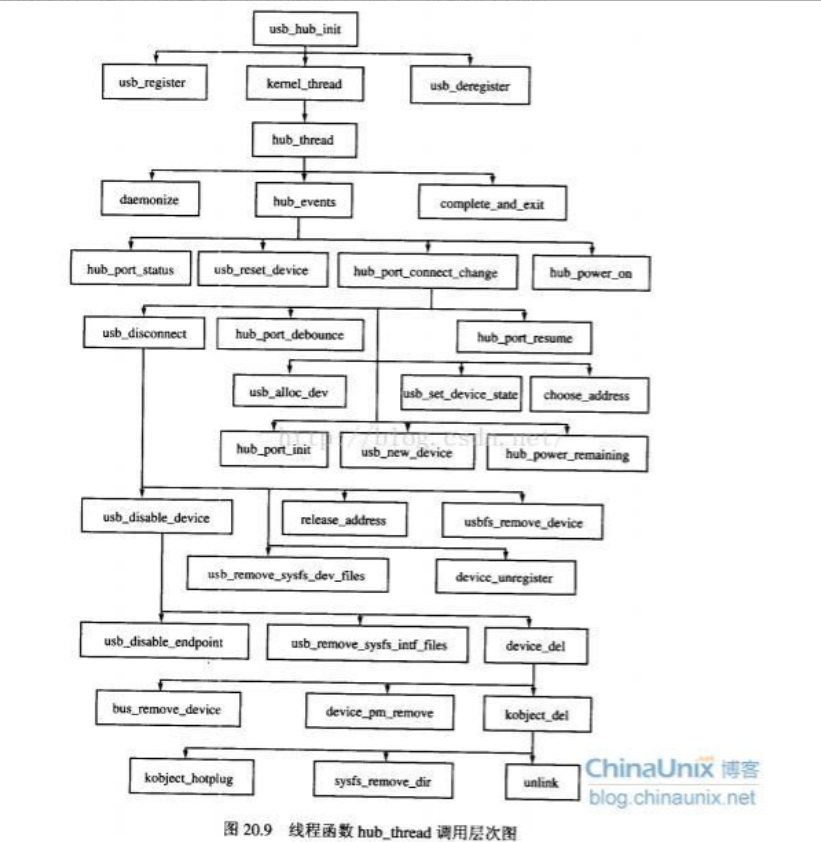
**3.软件层次分析-Hub部分热拔插**

前面主要是初始化的操作，也会去识别开机的时候已经插入的设备，如果是开机之后插入的设备，又是什么流程呢

前面是每当有设备连接到USB接口时，USB总线在查询hub状态信息的时候会触发hub的中断服务程序hub\_irq



这里我们同样贴出它的函数调用流程图,虽然是旧版本的，但是大致差不多（这里懒得自己画了，就剪切了个^^)



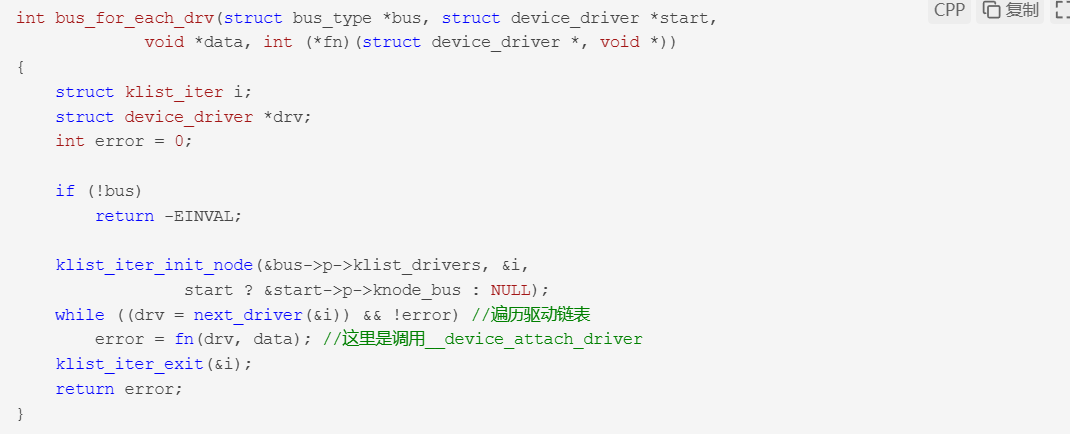
通过流程图我们可以清晰的明白，当usb设备插入usb接口后，hub\_irq执行，启动工作队列执行hub\_event工作，它检测到port状态的变化,调用hub\_port\_connect\_change(),如果是新设备那么usb\_allco\_dev，然后调用usb\_new\_device来进行配置使usb设备可以正常工作。

**二.usb驱动的probe匹配过程**

前面我们分析到调用usb\_new\_device来进行配置使usb设备可以正常工作，我们现在分析一下具体过程。主要是找到对应的客户驱动程序和该USB设备挂钩。

 usb\_new\_device中调用device\_add，将设备添加到设备层次结构中。

  大概调用流程：**device\_add -> bus\_probe\_device -> device\_initial\_probe -> \_\_device\_attach -> bus\_for\_each\_drv(dev->bus, NULL, &data, \_\_device\_attach\_driver);**



该函数调用bus\_for\_each\_drv()来从总线上已注册的所有驱动中找出匹配的驱动程序.遍历bus上的所有驱动程序,并为每个驱动调用fn()来查看是否匹配. 这里的fn就是\_\_device\_attach\_driver.这里化简了



**1. usb\_device\_match**

我们分析一下就是usb\_device\_match，这个函数只是做一些粗略的匹配, 如果匹配成功则返回1。这个函数只是做一些粗略的匹配, 如果匹配成功则返回1, 然后由driver\_probe\_device来做进一步的匹配, 如果匹配失败则返回0, 并且driver\_probe\_device也不会在执行. 这个函数的调用保证了dev, drv 要么都是设备级别的( 即dev 代表usb 设备,drv 代表usb 设备驱动), 要么都是接口级别的( 即dev 代表usb 设备的一个interface,drv 代表usb 接口驱动).



**2.driver\_probe\_device**

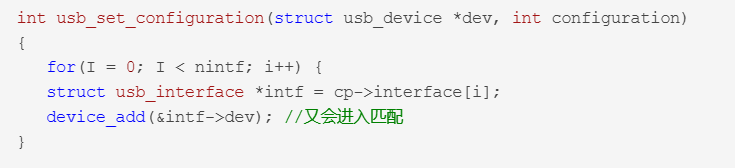
driver\_probe\_device主要是调用really\_probe -> (drv->probe)

对于usb 来说这个函数的调用有2 种分支, 1: dev,drv 代表的是设备级别的, 2 dev,drv 代表的是接口级别的. 其他情况组合在usb\_device\_match 中被过滤掉了.

**分支1: dev,drv 代表的是设备级别:**

此时的drv 肯定是usb\_generic\_driver. 因为在当前的usb 系统中只有这个driver 是代表整个设备的驱动, 它是在usb\_init 中被注册的, 而我们通常写的usb 驱动都是代表一个interface 的.

因此, 此时的drv->probe 将调用generic\_probe().再到usb\_set\_configuration

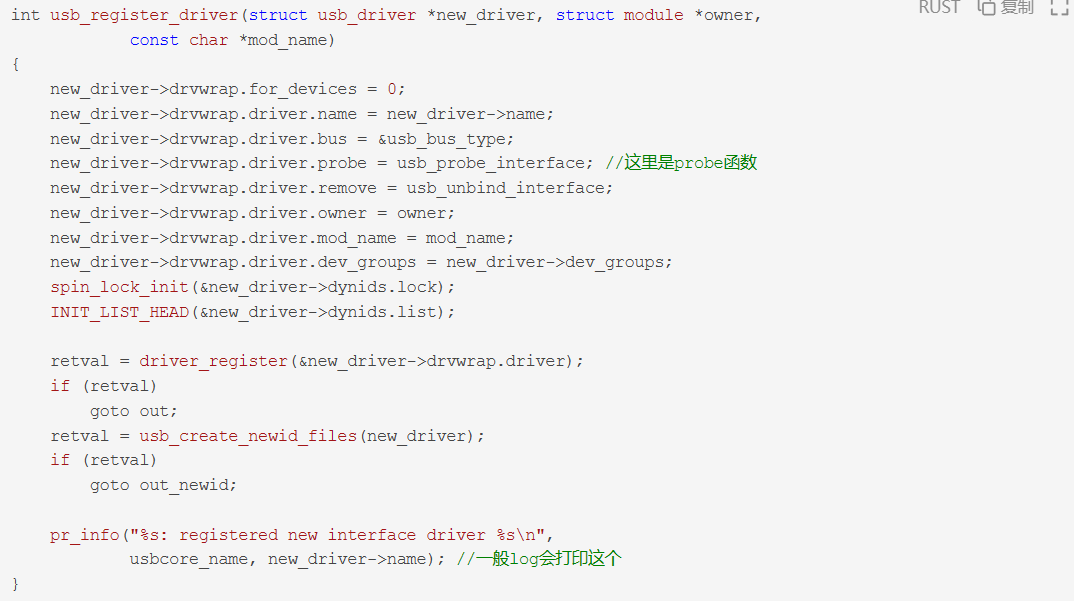


该函数比较重要, 但我们只关心probe 过程因此省掉了很多东西. 它为当前配置下的每个interface 调用device\_add() 函数, 根据前面的分析可知, 这个过程将会走到接下来我们要分析的分支2.

**分支2: dev,drv 代表的是interface 级别:**

此时的dev 代表着一个interface, 而drv 就代表了我们自己的usb 驱动. 但是我们应当看到drv是device\_driver 类型, 而我们写的usb 驱动的类型一般是usb\_driver, 因此这里的probe 和我们自己写的probe 显然不是同一个. 实际上这里的drv 是我们的驱动对象里内嵌的一个子对象( 因为linux 下所以的驱动都必须用device\_driver 来代表,). 那这个子对象的probe 函数是在哪里赋值的呢?

这就要看usb\_register宏了，实际里面是调用usb\_register\_driver



跟踪这个函数我们可以看到这里的probe 函数实际上是usb\_probe\_interface( 所有的usb interface 驱动都是一样的).



 driver->probe(intf, id); 这就调用到我们自己写的代码里面了,

**3.流程图**

大概流程图是一样的，我搬运来了

