# 前言 本文结构

第一章

第二章

第三章

第四章

第五章

第六章

# 第一章 USB整体初始化框架

用usb\_hub\_reset()函数清除掉所有之前集线器残留数据。

遍历设备数组usb\_dev[i]，将之前连接设备的残留数据擦除，并将其设备号usb\_dev[i].devnum设置为-1。

遍历控制寄存器（此处只有一个），先用usb\_lowlevel\_init（）函数进行底层设备初始化，对OHCI做初始化设置，并创建虚拟根集线器（USB总线）。

调用usb\_alloc\_new\_device（）函数将根集线器写入设备列表中第一个设备usb\_dev[0]。

调用函数usb\_new\_device（）扫描所有外部设备并将其添加到设备列表，以完成新建设备。

如果发生错误，调用usb\_free\_device（）擦除控制器数据以释放USB控制器。

其中，重点是usb\_lowlevel\_init（）、usb\_alloc\_new\_device（）与usb\_new\_device（）为重中之重。

# 第二章 USB底层初始化

其整体思路是：配置基础设置，复位OHCI，启动OHCI。

1 usb\_lowlevel\_init（指针，初始化类型，控制器）。获取OHCI地址，擦除开放主机控制器OHCI原有的数据。确保主机控制器通信区域HCCA是256字节对其的，获取HCCA地址，擦除HCCA原有数据。设置OHCI的状态为：挂起，不休眠、无中断请求、无BUG出现。初始化OHCI寄存器列表基地址0xBFE08000，设置主控制器接口名字为“OHCI”。调用hc\_reset(OHCI地址)以复位OHCI，调用hc\_start(OHCI地址)以启动OHCI。上述复位和启动失败时，调用hc\_release\_ohci()函数，擦除OHCI的数据。

2.1 hc\_reset(OHCI地址)，接管HC并成功复位。复位超时30ms，接管超时50ms。读寄存器OHCI寄存器列表中的控制寄存器ohci->regs->control，判断其第八位中断路由标志位是否为1，是的话写寄存器ohci->regs->cmdstatus请求变更所有权，最后循环读取控制寄存器状态检测是否在超时时间内完成了接管。通过写OHCI的中断禁止寄存器，禁止主机控制器HC的中断。同理，写OHCI的命令状态寄存器ohci->regs->cmdstatus，以完成在超时时间内复位。

2.2 hc\_start(OHCI地址)，配置OHCI相关寄存器与中断，创建虚拟根集线器。遍历OHCI下所有中断设备并初始化为“未连接”状态。通过读写寄存器操作告诉控制器控制列表和批量列表在哪里（现在列表是空的），设置帧计数器启动周期、间隔以及阈值。启动控制器操作，禁止所有中断并清除，打开我们需要的中断。创建虚拟根集线器：虚拟根集线器描述符a中描述无电源切换，虚拟根集线器状态寄存器中设置本地电源状态变化，最后将虚拟跟集线器设备号设置为0。

# 第三章 USB分配新设备

usb\_alloc\_new\_device（）此函数功能是向传入的指针指向设备数组中某个位置，并进行行相应的基础设置，完成分配一个新建的空白设备。如果设备数组的标号index等于最大连接数时，空间全部被占用，无法分配空白设备。设置其设备号，集线器端口数量，子设备，父设备，控制寄存器等。分配结束后index++，传入的指针指向设备数组中刚刚新建的设备。

# 第四章 USB新建新设备

本章小节结构图如下：

1

2.1

2.2

3.1

3.2

3.3

3.4

4.1

4.2

4.3

4.4

4.5

5

6

★

★

★

★

★

★

★

★

**新建USB设备框架**

注：带★的表示调用底层函数

首先声明，在此usb\_new\_device新建设备之前，usb\_alloc\_new\_device已经分配了新设备的存储地址以及某些相当基础的信息。

1 usb\_new\_device（USB设备），此函数只调用了usb\_setup\_device（）来创建设备，以及usb\_hub\_probe（）来探测新建的设备是不是集线器设备。两函数具体的功能如下。

2.1 usb\_setup\_device（USB设备，布尔变量read，父设备），这个布尔值永远为true。此函数的作用是，向已经分配好空间的USB设备写入他的详细信息，包括描述符等。该过程又分为两部分执行分别是usb\_prepare\_device（）准备USB设备，以及usb\_select\_config（）对设备进行更深层其的配置。

2.2 usb\_hub\_probe（USB设备，接口号），此函数下调用的usb\_hub\_check（）函数探测设备是否是集线器，如果不是集线器直接返回0，如果是集线器先调用函数usb\_hub\_configure（）对该集线器进行配置然后返回（配置不出错也是返回0）。

3.1 usb\_prepare\_device（USB设备，设备地址，布尔值read，父设备），通过usb\_setup\_descriptor（）来设置设备描述符中的最大传输数据包大小，通过usb\_set\_address（★）设置地址，本质就是调用usb\_control\_msg向下位机发送对应指令。

3.2 usb\_select\_config（USB设备），首先用get\_descriptor\_len（★）获取设备描述符，并用存储器复制memcpy将设备描述符从临时缓存区tmpbuf拷贝到dev->descriptor。用usb\_get\_configuration\_len（★）获取配置描述符长度，如果该长度大于0再利用usb\_get\_configuration\_no（★）获取配置描述符，并存储在临时缓存空间中。用usb\_parse\_config（）函数解析存储在数据缓存区中的配置描述符，解析后依次写入USB设备。用usb\_set\_maxpacket（）根据配置描述符设置每个端点最大传输数据包的大小。利用函数usb\_set\_configuration（★）向下位机发送对应命令更改其配置。最后清除残留的设备制造商信息、产品信息和产品序列号，重新写入上述信息。至此，当前USB设备握手成功。

3.3 usb\_hub\_check（USB设备，接口号），该函数识别USB设备是否是集线器，如果是集线器就返回0，不是集线器的话返回一个负值。判别方法用的是排除法，该接口的类iface->desc.bInterfaceClass如果不是集线器HUB类型的话排除，如果接口子类iface->desc.bInterfaceSubClass既不是0也不是1的话排除，如果该接口对应多个端点即iface->desc.bNumEndpoints不为一的话排除，如果该端点是输出端点的话排除，如果该端点不是中断端点的话排除，剩下的就是集线器了。

3.4 usb\_hub\_configure（USB设备），程序运行到此处已经能确定当前的USB设备是集线器了，所以我们要调用此函数对这个集线器进行配置。此配置分为以下三个方面：配置当前这个集线器相关信息，检查当前集线器配置信息，将此集线器加入扫描列表以扫描该集线器下挂载的更多USB设备。

首先，调用函数usb\_get\_hub\_device（）查看集线器设备数组hub\_dev[usb\_hub\_index]判断集线器的连接个数是否达到上限，如果没达到上限就分配一个设备空位给集线器设备hub，将当前集线器设备hub绑定其对应的USB设备。调用usb\_get\_hub\_descriptor（★）函数获取集线器设备描述符与其长度，将设备描述符指针descriptor指向数据缓存区buffer。在描述符叙述长度与描述符结构体实际长度间取最小值确定描述符实际长度，再次调用函数usb\_get\_hub\_descriptor（★）准确的获取设备描述符，并将获取到的设备描述符数据拷贝到hub->desc中。

接下来根据获取到的设备描述符配置集线器设备，如集线器特征，集线器位图，该集线器最大连接的子设备数量，配置集线器设备电源，判断该集线器是复合设备的一部分还是独立集线器，检测端口的过流保护，检测集线器设备描述符中的协议，配置集线器反应时间。

用函数usb\_get\_hub\_status（★）获取集线器状态，调用函数usb\_hub\_power\_on（）打开集线器电源。

用for循环根据此集线器子设备数量分配扫描设备空间，并添加到扫描设备列表。调用函数usb\_device\_list\_scan（）开始扫描集线器子设备。

4.1 usb\_setup\_descriptor（USB设备，布尔值），根据USB设备的速度来判断它是低速设备、高速设备还是全速设备。如果是低速和高速设备就按照其速度，直接配置其最大传输包的大小dev->descriptor.bMaxPacketSize0，并将其写入对应的输入输出端点dev->epmaxpacketin\out[0]中，最后对此大小进行编码，写入dev->maxpacketsize中方便计算。如果是全速设备就调用get\_descriptor\_len（★）函数获取最大数据包大小后设置。

4.2 usb\_parse\_config（USB设备，数据缓存区），解析位于缓冲区中的配置，并填充 dev->config 结构。先将数据缓存区全部的配置描述符数据拷贝到dev->config中，然后对数据缓存区的数据进行分段解析，如果发现是新的接口数据就将他拷贝到if\_desc，如果是新的端点数据就将他拷贝到if\_desc->ep\_desc[epno]。

4.3 usb\_set\_maxpacket（USB设备），设置给定配置中所有端点的最大打包值。两层for循环，第一层遍历该设备所有接口dev->config.desc.bNumInterfaces，第二层遍历对应接口下所有端点dev->config.if\_desc[i].desc.bNumEndpoints，多次调用函数usb\_set\_maxpacket\_ep（USB设备，接口号，端点号）。在该函数中，根据端点性质ep->bmAttributes计算出该端点是控制端点、输入端点还是输出端点，最后向相应位置赋值最大打包值。

4.4 usb\_hub\_power\_on（USB集线器设备），首先从USB集线器设备找到对应的USB设备。遍历所有子设备，用usb\_set\_port\_feature（★）函数使能子设备电源。设置延时时间值以便在查询设备之前其电源稳定工作，并设定开机超时时间值。

4.5 usb\_device\_list\_scan（无参数）扫描集线器设备下的子设备。循环进行如下过程：首先调用函数list\_empty（）查看当前集线器下的扫描设备列表是否为空，如果为空代表所有子设备遍历完成，可以结束对当前集线器的遍历。否则的话，先用list\_for\_each\_entry\_safe（）函数检查当前扫描设备列表入口是否是符合规范的，然后进一步用usb\_scan\_port（）开始对特定子设备开始扫描，直到扫描不到设备结束循环。

5 usb\_scan\_port（扫描设备），首先加载USB设备、集线器设备以及扫描端口。先用usb\_get\_port\_status（★）函数获取端口状态，如果获取失败就从扫描列表删除此设备。从上述端口状态读取端口连接变化信息与设备连接状态信息，如果端口连接无变化且无设备连接说明是空集线器，从扫描列表删除此设备。端口复位与端口超速就调用usb\_clear\_port\_feature（★）清除端口特性。

到此说明集线器下有子设备连接，且新设备已经准备就绪。调用usb\_hub\_port\_connect\_change（）函数改变端口连接。清除端口特性后检查有没有子设备连接，如果有的话再次调用usb\_hub\_port\_connect\_change（）函数。

最后检查状态：端口状态为暂停就清除端口特性，电流过载状态就清除端口特性并重新设置。如果未达到最大扫描计数，则返回而不从扫描列表中删除设备，这将重新发出新的扫描，扫描子设备的子设备。否则我们已经完成了这个子设备的扫描，所以让我们从扫描列表中删除这个设备并返回。

6 usb\_hub\_port\_connect\_change（USB设备，端口），这个函数是重中之重，就是靠它实现了集线器扫描的核心。先调用函数usb\_get\_port\_status（★）获取端口状态，再调用usb\_clear\_port\_feature（★）清除端口特性，根据端口特性判断该端口下是否有连接，如没有任何连接立即返回。

确认有连接后调用usb\_hub\_port\_reset（★）复位端口，根据端口状态配置通信速度。最后掉用函数usb\_alloc\_new\_device（**递归**）分配新设备空间，配置父设备子设备端口关系实现连接，调用函数usb\_new\_device（**递归**）新建设备。

# 第五章 提交命令到最底层

本章小节结构图如下：

1

**底层**

**核心**

**程序**

**框架**

2

3

4

8

5.1

5.2

6.1

6.2

6.3

6.4

7.1

7.2

7.3

7.4

本章从相对底层的usb\_get\_descriptor函数（）出发查看代码执行过程。

1 usb\_get\_descriptor(USB设备，类型，指针，数据缓存区地址，大小)，其中只调用了一个函数usb\_control\_msg(USB设备，管道，请求，请求类型，值，指针，数据缓存区地址，大小，超时时间值)。**管道**由设备和端点0计算得出，其中包含了设备号，端点号，最大传输数据大小以及传输方向。**请求**为获取描述符USB\_REQ\_GET\_DESCRIPTOR。**请求类型**为输入USB\_DIR\_IN。值为(type << 8) + index。指针为0。**超时时间值**为100ms。【拓展请求命令，创建管道】

2 usb\_control\_msg(USB设备，管道，请求，请求类型，值，指针，数据缓存区地址，大小，超时时间值)。此函数下创建了一个名为setup\_packet的命令包，其类型为struct devrequest，将请求类型、请求、设置值、指针与大小打包放入setup\_packet之中。将设备状态设置为未处理后，调用submit\_control\_msg（USB设备，管道，数据缓存区地址，传输长度，命令包）提交控制信息。之后在超时时间内循环查看设备状态，如果设备状态不再是未处理的就结束循环，最后查看跳出循环是否因为超时引起的，超时的话就返回错误代码否则返回接收数据包的长度。【将命令打包，超时监测】

3 submit\_control\_msg（USB设备，管道，数据缓存区地址，传输长度，命令包）。此函数只调用了\_ohci\_submit\_control\_msg(**开放的主机控制器接口OHCI的地址**,USB设备,管道, 数据缓存区地址，传输长度，命令包)函数，达到控制OHCI底层。【调用开放的主机控制器接口】

4 \_ohci\_submit\_control\_msg(开放的主机控制器接口OHCI的地址，USB设备，管道，数据缓存区地址，传输长度，命令包)。根据管道中的信息[第一步]解码出该管道的最大传输大小，并检测其是否为0。检测管道中的设备号[第一步]是否是根集线器的设备号。如果是就进行根集线器的重定向：OHCI中保存的根集线器设备设置为当前的USB设备，再调用ohci\_submit\_rh\_msg(OHCI地址，USB设备，管道，数据缓存区地址，传输长度，命令包)。否则调用submit\_common\_msg(OHCI地址，USB设备，管道，数据缓存区地址，传输长度，命令包，间隔)访问其他公共外设，此处间隔为0。【检验参数最大传输大小，判断管道连接的设备是否为根集线器，OHCI访问设备】

5.1 ohci\_submit\_rh\_msg(OHCI地址，USB设备，管道，数据缓存区地址，传输长度，命令包)。开辟一个名为databuf的数据缓存空间，调用usb\_pipeint对管道进行初始化（将管道设置为中断类型）。从命令包解码出请求类型和请求、值、指针、长度，根据解码出的请求进行相应配置：①如为获取状态则根据需要获取的具体信息（接口信息、端点信息、类信息等）将databuf指针指向相应值或位置，并对长度进行赋值；②如为清除指令，则根据请求类型和设置值（电流过载，端口使能，端口停止，端口供电，端口连接，端口复位等）对相应寄存器ohci->regs->roothub.status赋值，返回数据长度为0无返回数据；③如果为设置指令也是根据设置信息对相关寄存器赋值（端口停止、端口复位、端口供电、端口使能以及地址设置将根集线器设备号设置为wValue），返回数据长度0；④如为获取描述符（设备描述符，配置描述符，字符串描述符等）就将数据缓存区databuf的指针指向本地存储的根集线器描述符，并返会该描述符的长度；⑤如为获取根集线器类描述符就将各种信息依次填入数据缓存区databuf中，计算返回数据的长度；⑥如为获取配置与设置配置同上。进行完上述操作后，计算数据缓存区databuf中有用数据的长度并将其写入USB设备的接收应答长度dev->act\_len中，利用memcpy函数将有用的信息从数据缓存区databuf中拷贝到我们传入的缓存区地址下，最后更新USB设备的状态为start。【解码命令包，根据命令读取相应信息到缓存区并计算返回数据长度或根据命令信息配置相关寄存器，将缓存区的数据拷贝到我们想要的地址下，更新USB设备的状态dev->status与接收长度dev->act\_len】

5.2 submit\_common\_msg(OHCI地址，USB设备，管道，数据缓存区地址，传输长度，命令包，间隔)。从管道pipe中解码出数据包大小的最大值，确保其大于0。调用ohci\_alloc\_urb函数配置USB请求块urb（USB设备，管道，数据缓存区地址，传输长度，间隔）。用ohci\_get\_ohci\_dev（OHCI地址，USB设备的设备号，管道中断）查找OHCI设备。调用sohci\_submit\_job（OHCI地址，OHCI设备，USB请求块，命令包）向下位机发送开始传输的命令，用USB\_TIMEOUT\_MS函数提取管道中包含的超时时间值[第一步]，在超时时间内循环用hc\_interrupt函数监测主机控制器中断状态，以此判断传输知否结束，正常结束时传递状态为USB\_ST\_CRC\_ERR，如果超时状态设置为USB\_ST\_CRC\_ERR。最后，更新设备状态dev->status与返回长度dev->act\_len，如果状态异常则清除数据缓存区buffer，释放USB请求块urb。【配置USB请求块，查找OHCI设备，向下位机发送开始传输的命令，检测中断是否超时，更新USB状态与接收数据的长度，释放请求块urb】

6.1 ohci\_alloc\_urb(USB设备，管道，数据缓存区地址，传输长度，间隔)。创建一个类型为urb\_priv\_t的USB请求块urb，将USB设备、管道、数据缓存区地址，传输长度以及间隔统统打包到USB请求块中。【将USB设备、管道、数据缓存区地址，传输长度以及间隔统统打包到USB请求块中】

6.2 ohci\_get\_ohci\_dev(OHCI地址，USB设备号，管道中断)，首先根据USB设备号在OHCI的设备列表中查找OHCI之前是否与此设备建立连接(ohci->int\_dev[i].devnum == devnum)，如果有返回对应的OHCI设备地址&ohci->int\_dev[i]。如果没有建立连接就找一个空位置新建设备并返回。否则即找不到设备，又没有空位置返回NULL。【根据设备号在OHCI设备列表中查找、新建OHCI设备】

6.3 sohci\_submit\_job(OHCI地址，OHCI设备，USB请求块，命令包)。从USB请求块中解码出USB设备，管道，数据缓存区地址，传输长度和间隔。检测OHCI控制器是否挂起，将请求块完成标志urb->finished置为0表示未完成。调用ep\_add\_ed（）函数创建端点ed。根据管道类型设置数据传输区间TD的个数size（批量传输管道和控制传输管道单次传输4096个字节，中断传输一个TD即可）。对端点USB请求块的私有数据purb\_priv结构体赋值。用td\_alloc（）分配数据传输区间TD，如果端点还没有运行，则调用ep\_link（）将端点ed链接到主机控制器HC中，最后td\_submit\_job（）准备传输所有数据！！！【创建端点，计算数据传输区间TD的个数并进行分配，链接端点，准备开启传输】

6.4 hc\_interrupt(OHCI地址)，检查主机控制器中断触发原因，进行状态检测，判断传输是否顺利结束。确保OHCI与HCCA状态正常，否则返回错误代码。如果是根集线器状态变更，返回0xFF（对通用USB设备来说，这是错误代码之一）。中断发生不可描述的错误时，复位主机控制器HC和总线，返回错误代码-1。调度超限返回错误代码-1。完成了头回写的话读写寄存器regs->intrdisable并调用函数dl\_done\_list（）进行删除已经完成的TD等相关操作。【监测传输状态】

7.1 ep\_add\_ed(OHCI设备，USB设备，管道，间隔，加载)创建与初始化端点（参数加载为1）。定向创建端点，让端点指针ep定向指向数组ohci\_dev->ed[]中的某个端点，以完成创建。如果端点是新建端点就初始化，创建第一个数据传输区间td[0]。设置新建端点的状态：尾指针，头指针，端点状态，管道类型，信息，中断，加载，OHCI设备当前端点。【创建端点，并初始化其状态】

7.2 td\_alloc(OHCI设备，USB设备)，循环调用size次函数来分配size个数据传输区间TD。在对应的ohci\_device下的td[]中找一个空节点（未被USB设备占用的td），返回该空节点地址，让purb\_priv->td[i]指针指向它，以完成向设备分配TD的过程。【查找未被分配的td进行分配】

7.3 ep\_link(OHCI地址，端点)，根据端点类型将端点链接到主机控制器列表中。【链接端点】

7.4 td\_submit\_job(OHCI地址，USB设备，管道，数据缓存区地址，传输长度，命令包，请求块，间隔)。复位USB切换位（OHCI自己处理数据切换，我们只使用 USB 切换位进行重置即可）。根据管道类型，调用td\_fill（）函数配置每个TD（本质是将数据缓存区buffer一小段一小段得分配给各个TD）。调用ohci\_writel（）函数，向寄存器ohci->regs->cmdstatus写入命令开始数据传输。【复位数据切换位，配置TD，开启数据传输】

8 td\_fill(OHCI地址，信息，数据缓存区起始地址，长度，USB设备，指针，请求块)。设置端点，接收数据缓存区，信息，当前缓存区指针，缓存区结束地址，下一个TD，尾指针。【设置TD】

# 第六章 停止USB设备

1 usb\_stop（）函数停止USB设备的使用，先调用usb\_hub\_reset（）进行集线器复位，再多次调用usb\_lowlevel\_stop（）停止USB底层设备。

2.1 usb\_hub\_reset（）将全局集线器设备数组hub\_dev[ ]清零以防再次使用。

2.2 usb\_lowlevel\_stop（设备在设备数组中的位置）调用函数hc\_reset（）复位OHCI设备以及总线，该函数详细信息在第二章中的2.1小节。