一、 目录

二、 相关代码

1. USB 三种返回确认包（状态包）

三种返回确认信息

ACK 、NAK 、STALL

1.1 【ACK 包】

ACK(确认) 表示 主机和设备已经收到数据，没有出现错误。设备必须在 Setup 事务的交换包中返回 ACK，设备也必须在 OUT 事务的交换中返回 ACK。主机在 IN 事务的交换包中返回 ACK。

1.2 【NAK 包】

（NAK 包只能从设备发向主机）

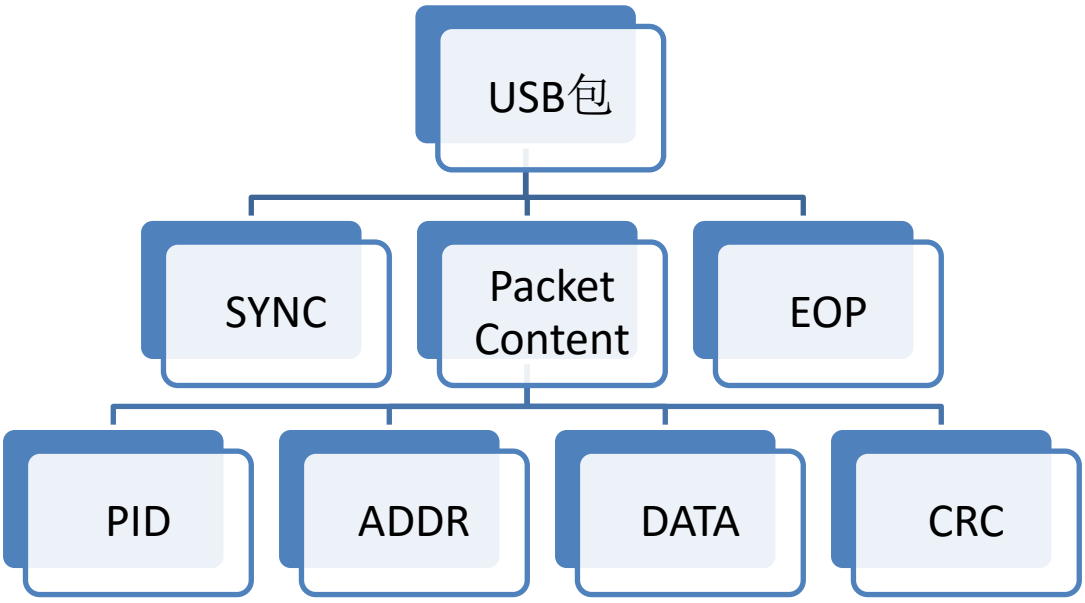
NAK(未确认) 表示设备正忙或没有数据要返回。如果主机在设备太忙而不能接受数据时发送数据，则设备在交换包中发出 NAK。如果主机在设备没有数据可发送时向设备请求数据，则设备在数据包中发出 NAK。在上述两种情况的任何一种下，NAK 表示一个暂时的状况，而主机会在以后重试。

1.3 【STALL 包】

不支持的控制请求，控制请求失败或终端失败 当一个设备接收到一个终端不支持的控制传输请求，那么这个设备返回一个 STALL 给主机。设备在它支持这个请求但是由于某些原因不能采取请求的动作时，也会发出 STALL 给主机。 STALL 的另一个用途是在终端暂停特性设置的情况下来响应传输请求，表示终端根本不能发送或接收数据。规范称这个类型的延迟为功能延迟。批量和中断终端必须支持功能延迟。在接收到一个功能 STALL 后，主机停止所有与设备悬而未决的请求，并且不会恢复通信直到它已经发送一个成功的请求来清除设备的暂停特性。主机绝不会发送 STALL

2. USB 数据包分类及说明

1.1 组成



说明：
SYNC:同步字段
PID：包标识符
ADDR：地址字段（设备地址+端口地址）
DATA：数据字段，包括帧号
CRC：校验字段
EOP：包结束

1.2 包标识符(PID)

在 USB 协议中，USB 有很多不同类型的包，通过 PID 来区分

PID 类型	PID 名	PID[6..0]	描述
令牌包 (Token)	输出（OUT）	0x87	Host To Device 包中有地址+端口号
	输入（IN）	0x5A	Device To Host 包中有地址+端口号
	帧起始(SOF)	0xA5	Host To Device 帧开始标记和帧号
	建立(Setup)	0XB4	Host To Device 建立一个控制管道的事务包数据有地址+端口号
数据包（DATA）	数据 0(DATA0)	0xC3	偶数据包
	数据 1(DATA1)	0xD2	奇数据包
握手包（HandShake）	确认(ACK)	0x4B	接收无错误的响应
	不确认(NAK)	0x5A	接收错误或者忙的响应
	停止(STALL)	0x1E	设备出错或者协议不支持的响应
专用包 (Special)		0x2C	主机发送的前同步字，由来区分低速和高速设备

1.3 地址字段（ADDR）11Bit

地址字段(ADDR)由设备地址和端点号组成

设备地址 7bit：

LSB						MSB
Addr0	Addr1	Addr2	Addr3	Addr4	Addr5	Addr6

端点地址 4bit:

LSB			MSB
Endp0	Endp1	Endp2	Endp3

1.4 Packet 分四大类

命令：token packet

帧首：start of frame

数据：data

握手：handshake

令牌包(token packet)

令牌包(token packet)					
Sync	In/Out/Setup	ADDR	ENDP	CRC5	EOP
00000001	0xB4	3	0	0x0A	250ns

帧起始（start of frame）

帧起始（start of frame）				
Sync	SOF	Frame	CRC5	EOP
00000001	0xA5	1611	0x11	250ns

数据包（DATA）

数据包（DATA）				
Sync	Data0	Data	CRC16	EOP
00000001	0xC3	80 06 00 01 00 00 12 00	0x072F	250ns

确认包/握手包（handshake）

确认包/握手包（handshake）		
Sync	ACK	EOP
00000001	0x4B	233ns

三、 USB 设备协议

1. 内容：

USB 描述符
USB 设备命令
USB 设备的状态图
USB 总线枚举

2. USB 设备枚举过程

当一个 USB 设备接入后，会有以下执行过程

1. 供电
2. 复位
3. 获取Device Descriptor
4. 复位（可选）
5. 分配地址
6. 获取Device Descriptor
7. 获取Configuration Descriptor
8. 获取String Descriptor(可选)
9. 配置

3. USB 描述符

通过一套描述符，USB 设备向 USB 主机描述自己的功能、属性、配置等信息

作用：描述符的作用在于设备向主机汇报自己的信息、特征，主机根据这些信息从而加载相应

分类：描述符分为三大类：标准描述符、设备类描述符、厂商描述符。

1.1 描述符类型说明 bDescriptorType

类型	值	描述符
标准描述符	0x01	设备描述符（Device Descriptor)
	0x02	配置描述符（Configuration Descriptor）
	0x03	字符串描述符（String Descriptor）
	0x04	接口描述符（Interface Descriptor）
	0x05	端点描述符（EndPont Descriptor）
	0x06	设备限定描述符
	0x07	其他速率配置描述符
类描述符	0x29	集线器类描述符（Hub Descriptor）
	0x21	人机接口类描述符（HID）
厂商定义	0xFF	
组合设备	0x0B	IAD 描述符类型 USB 复合设备一般用 Interface Association Descriptor（IAD）实现，就是在要合并的接口前加上 IAD 描述符。

1.2 设备描述符(Device Descriptor)(0x01)

序号	域	大小	值	描述
0	bLength	1	数字	描述设备描述符的总字节数
1	bDecriptorType	1	常量	描述符的类型（为 0X01，这里是设备描述符）
3	bcdUSB	2	BCD	这个设备兼容的 USB 设备版本号
4	bDeviceClass	1	类型	设备类码：是由 USB 协会规定的，描述的是接口所能实现的功能。当此域为 0 时下面的子类也必须为 0，当为 0XFF 表示的是厂商自定义设备类
5	bDeviceSubClass	1	子类	子类代码码：这个码值的意思是根据设备类码来看。如设备类码为零，这字段也要零，如设备类码为 0XFF，此域的所有值保留。
6	bDevicePortocol	1	协议	协议码：这些码的值视设备码和子类代码的值而定。当该字段为 0 是，表示设备不使用类所定义的协议， 当该字段的值为 0XFF 时，表示使用设备厂商自定义的协议
7	bMaxPacketSize0	1	字数	端点 0 的能缓冲的最大数据包大小
8	idVendor	2	ID	生产设备厂家的标志（由 USB 相关组织给的）
10	idProduct	2	ID	产品标志（由生产的厂家自己做编号）
12	bcdDevice	2	BCD	设备的版本号
14	iManufacturer	1	索引	描述生产设备厂家的信息的字符串描述符的索引值。
15	iProduct	1	索引	描述所使用设备产品的信息的字串描述符的索引值。
16	iSerialNumber	1	索引	描述设备产品的序列号信息的字串描述符的索引值。
17	bNumConfigurations	1	数字	设备有多少种配置

【说明 1:】当设备类型 bDeviceClass = 0 时，说明类型将由接口描述符中定义的为准。

【说明 2:】从设备描述符表格中可知，有 3 个索引值：厂商信息索引、产品信息索引、设备序列号索引，这意味着，将有 3 个字符串描述符为其准备。

设备描述符举例说明：

```
const u8 Virtual_Com_Port_DeviceDescriptor[] =       //设备描述符--数组
  {
    0x12,           /* bLength */  //整个描述符长度--0x12/18 个字节
    0x01,           /* bDescriptorType */              //类别--0X01 设备描述符
    0x00, 0x02,     /* bcdUSB = 2.00 *///此设备与描述表兼容的 USB 设备说明版本号（BCD
    码）
    0xEF,           /* bDeviceClass: CDC */ //设备类码：0X02 CDC 控制类
    0x02,           /* bDeviceSubClass */   //子类挖码
    0x01,           /* bDeviceProtocol */   //协议码
    0x40,           /* bMaxPacketSize0 */   //端点 0 的最大包大小（仅 8,16,32,64  为合法值）
    0x83, 0x04,     /* idVendor = 0x0483 */  //厂商标志（由 USB-IF 组织赋值）
    0x40, 0x57,     /* idProduct = 0x7540 */     //产品标志（由厂商赋值）
    0x00 0x02,      /* bcdDevice = 2.00 */   //设备的版本号
    1,              /* Index of string descriptor describing manufacturer */
                    //描述厂商信息的字符串描述符的索引值。
    2,              /* Index of string descriptor describing product */
                    //描述产品信息的字串描述符的索引值。
    3,              /* Index of string descriptor describing the device's serial number */
                    //描述设备序列号信息的字串描述符的索引值。
    0x01            /* bNumConfigurations */
                    //可能的配置描述符数目 1
  };
```

1.3 配置描述符(Configuration Descriptor)(0x02)

配置描述符中包括了描述符的长度(属于此描述符的所有接口描述符和端点描述符的长度的和)、供电方式(自供电/总线供电)、最大耗电量等。主果主机发出 USB 标准命令 Get Descriptor 要求得到设备的某个配置描述符，那么除了此配置描述符以外，此配置包含的所有接口描述符与端点描述符都将提供给 USB 主机。

序号	域	大小	值	描述
0	bLength	1	数字	此描述表的字节数长度。
1	bDecriptorType	1	常量	配置描述表类型（此处为 0x02）
2	wTotalLength	2	数字	此配置信息的总长（包括配置，接口，端点和设备类及厂商定义的描述符）
4	bNumInterfaces	1	数字	此配置所支持的接口个数
5	bCongfigurationValue	1	数字	在 SetConfiguration（）请求中用作参数来选定此配置。
6	iConfiguration	1	索引	描述此配置的字串描述符的索引
7	bmAttributes	1	位图	配置特性： D7： 保留（设为一） D6： 自给电源 D5： 远程唤醒 D4..D0：保留（设为一） 一个既用总线电源又有自给电源的设备会在 MaxPower 域指出需要从总线取的电量。并设置 D6 为一。运行时期的实际电源模式可由 GetStatus(DEVICE) 请求得到。
8	MaxPower	1	mA	在此配置下的总线电源耗费量。以 2mA 为一个单位

【说明 1:】配置描述符也包含了个用于描述符该配置的字符串描述符索引 iConfiguration，这说明将有个字符串描述符为其准备。

【说明 2:】枚举的过程可分为 4 个状态阶段：接入状态阶段、缺省状态阶段、地址状态阶段、设置状态阶段，各状态阶段任务如下：

接入状态阶段----------主机检测到新设备接入后，将复位总线（释放总线于空闲状态）。

缺省状态阶段----------主机利用 0x00 地址访问新接入的设备，读取部分描述符后，会分配个设备地址。

地址状态阶段----------主机再次复位总线，然后用新分配的地址获取设备所有的描述符。

设置状态阶段----------主机根据设备的描述符，会对设备作些相关的配置。

【说明 3:】bCongfigurationValue----------USB 设备的配置值。用于存放主机执行 SetConfiguration 命令的设置值。当主机发送 GetConfiguration 命令时，设备将向主机返回 1 个字节的配置值。然而，USB 设备处于不同的状态时，对 GetConfigration 的请求也有不同的响应：

1.> 在枚举阶段，若设备处于地址状态时，对 GetConfigration 的请求返回为 0；

2.> 在枚举阶段，若设备处于默认状态(缺省状态)时，对 GetConfigration 的请求视为无

效；

3.> 在枚举阶段，若设备处于配置状态时，对 GetConfigration 的请求将返回 bConfigurationValue 字段的值(该值可能是配置描述符的默认值，也可能是 USB 主机的设置值，这要看在执行 GetConfigration 命令前是否执行了 SetConfigration 命令)。

因为主机要执行 SetConfigration 命令，所以 bCongfigurationValue 的默认值没什么用。实际上主机给 bCongfigurationValue 赋值后，bCongfigurationValue 值就充当配置描述符的编号，用以区分不同的配置，因为一个设备可能有多个配置。

设备描述符举例说明：

```
const u8 Virtual_Com_Port_ConfigDescriptor[] =          //配置描述符
  {
    0x09,          //整个描述符长度--0x09/9 个字节
    0x02,          //类别--0X02 设备描述符
    0x67,0x00      //此配置信息的总长（包括配置，接口，端点和设备类及厂商定义的描述符）
    0x02,     //此配置所支持的接口个数
    0x01,     //在 SetConfiguration（）请求中用作参数来选定此配置。
    0x00,     //描述此配置的字串描述表索引 0
    0xC0,     //配置特性：D7： 保留（设为一），D6： 自给电源，D5： 远程唤醒，D4..0：
              保留（设为一）
    0x32,      //在此配置下的总线电源耗费量。以 2mA 为一个单位。
 };
```

1.4 接口描述符(Interface Descriptor)(0x04)

接口：接口用于描述特定的功能每个接口都有一个端点集，用于实现接口功能
配置描述符中包含了一个或多个接口描述符，这里的“接口”并不是指物理存在的接口，在这里把它称之为“功能”更易理解些，例如一个设备既有录音的功能又有扬声器的功能，则这个设备至少就有两个“接口”。
如果一个配置描述符不止支持一个接口描述符，并且每个接口描述符都有一个或多个端点描述符，那么在响应 USB 主机的配置描述符命令时，USB 设备的端点描述符总是紧跟着相关的接口描述符后面，作为配置描述符的一部分被返回。接口描述符不可直接用 Set_Descriptor 和 Get_Descriptor 来存取。
如果一个接口仅使用端点 0，则接口描述符以后就不再返回端点描述符，并且此接口表现的是一个控制接口的特性，它使用与端点 0 相关联的默认管道进行数据传输。在这种情况下 bNumberEndpoints 域应被设置成 0。接口描述符在说明端点个数并不把端点 0 计算在内。

序号	域	大小	值	描述
0	bLength	1	数字	此表的字节数
1	bDecriptorType	1	常量	接口描述表类（此处应为 0x04）
2	bInterfaceNumber	1	数字	接口号，当前配置支持的接口数组索引（从零开始）。
3	bAlternateSetting	1	数字	可替换的接口描述符编号。实际就是接口的描述符的编号。
4	bNumEndpoints	1	数字	该接口使用的端点数，不包括端点 0
5	bInterfaceClass	1	类	接口所属的类值： 零值为将来的标准保留。 如果此域的值设为 FFH，则此接口类由厂商说明。所有其它的值由 USB 说明保留。
6	bInterfaceSubClass	1	子类	子类码 这些值的定义视 bInterfaceClass 域而定。如果 bInterfaceClass 域的值为零则此域的值必须为零。bInterfaceClass 域不为 FFH 则所有值由 USB 所保留。
7	bInterfaceProtocol	1	协议	协议码：bInterfaceClass 和 bInterfaceSubClass 域的值而定。如果一个接口支持设备类相关的请求此域的值指出了设备类说明中所定义的协议。
8	iInterface	1	索引	描述此接口的字串描述表的索引值。

【说明 1：】接口描述符中用到接口编号 bInterfaceNumber，以区分在同一配置下的不同的接口。同时还有该接口描述符的索引 iInterface，这意味着将为其准备准备一个字符串描述符。

【说明 2：】接口描述符中有一项：可替换的接口描述符编号 bAlternateSetting，表示对某一接口进行描述的描述符编号。虽然，USB 设备的配置与配置描述符是一一对应的，即一个配置只能由一个配置描述来描述它，但一个接口却允许有多种描述符来描述它，尽管接口描述符的编号还是唯一一个。说白了就是：一个接口有唯一的一个接口编号，但一个接口却可以有多个不同的描述符编号，而这些不同的接口描述符的编号值就是 bAlternateSetting。所以，通过 bInterfaceNumber 可以选定一个唯一的接口，然后再通过

bAlternateSetting 选择想要的对该接口的描述。主机通过 GetInterface 可以获取当前正在使用的接口及接口描述，通过 SetInerface 可以选定某接口及其使用的描述符。

接口描述符举例说明：

0x09,　　//整个描述符长度--0x09/9 个字节
0x04,　　//接口描述表类（此处应为 0x04）
0x00,　　//接口号，当前配置支持的接口数组索引（从零开始）。
0x00,　　//可选设置的索引值。
0x01,　　//此接口用的端点数量，如果是 0 则说明此接口只用缺省控制管道。
0x02,　　//接口所属的类值：0 值为将来的标准保留。 如果此域的值设为 FFH，则此接口类由厂商说明。 所有其它的值由 USB 说明保留。,0x02 CDC 控制类
0x02,　　//子类码:这些值的定义视 bInterfaceClass 域而定。 如果 bInterfaceClass 域的值为零则此域的值必须为零。 bInterfaceClass 域不为 FFH 则所有值由 USB 所保留。
0x01,　　//协议码：bInterfaceClass 和 bInterfaceSubClass 域的值而定.如果一个接口支持设备类相关的请求此域的值指出了设备类说明中所定义的协议.
0x00,　//描述此接口的字串描述表的索引值。

1.5 端点描述符(Endpoint Descriptor)(0x05)

端点是设备与主机之间进行数据传输的逻辑接口，除配置使用的端点 0（控制端点，一般一个设备只有一个控制端点）为双向端口外，其它均为单向。端点描述符描述了数据的传输类型、传输方向、数据包大小和端点号（也可称为端点地址）等。
除了描述符中描述的端点外，每个设备必须要有一个默认的控制型端点，地址为 0，它的数据传输为双向，而且没有专门的描述符，只是在设备描述符中定义了它的最大包长度。主机通过此端点向设备发送命令，获得设备的各种描述符的信息，并通过它来配置设备。

序号	域	大小	值	描述
0	bLength	1	数字	此表的字节数
1	bDecriptorType	1	常量	端点描述表类（此处应为 0x05）
2	bEndpointAddress	1	端点	此描述表所描述的端点的地址、方向： Bit 3..0：端点号. Bit 6..4：保留,为零 Bit 7: 方向,如果控制端点则略。 0：输出端点（主机到设备） 1：输入端点（设备到主机）
3	bmAttributes	1	位图	此域的值描述的是在 bConfigurationValue 域所指的配置下端点的特性。 Bit 1..0 :传送类型 00=控制传送 01=同步传送 10=批传送 11=中断传送 所有其它的位都保留。
4	wMaxPacketSize	2	数字	当前配置下此端点能够接收或发送的最大数据包的大小。对于实进传输，此值用于为每帧的数据净负荷预留时间。在实际运行时，管道可能不完全需要预留的带宽，实际带宽可由设备通过一种非 USB 定义的机制汇报给主机。对于中断传输，批量传输和控制传输，端点可能发送比之短的数据包
6	bInterval	1	数字	周期数据传输端点的时间间隙。 此域的值对于批传送的端点及控制传送的端点无意义。对于同步传送的端点此域必需为 1，表示周期为 1ms。对于中断传送的端点此域值的范围为 1ms 到 255ms。

【说明 1：】端点的传输类型字节 bmAttributes，描述了该端点的传输特性：0~1bit 定义了传输类型---------00=控制传输、01=同步传输、10=批量传输、11=中断传输。

【说明 2：】周期端点的访问周期字节 bInterval，定义了该端点被主机的访问周期，此域值对于批量传输和控制传输毫无意义。对于同步传输，其值必须为 1，即 1ms 为标准的同步帧周期。对于中断传输，该值为 1~255，即 1ms~255ms。

端点描述符举例说明

0x07,　　//描述符长度：7 字节

0x05,　　//端点描述类型 0x04

0x82,　　//端点：此描述表所描述的端点的地址、方向：

//Bit 3..0 : 端点号，

//Bit 6..4 : 保留,为零，

//Bit 7: 方向,如果控制端点则略。0：输出端点（主机到设备），1：输入端点（设备到主机）

0x03,　　//传输类型：Bit 1..0 :传送类型；00=控制传送，01=同步传送，10=批传送，11=中断传送

0x08,0x00　　/* wMaxPacketSize: */　　//端点能够接收或发送的最大数据包的大小。对于实进传输，此值用于为每帧的数据净负荷预留时间。在实际运行时，管道可能不完全需要预留的带宽，实际带宽可由设备通过一种非 USB 定义的机制汇报给主机。对于中断传输，批量传输和控制传输，端点可能发送比之短的数据包

0xFF,　　//周期数据传输端点的时间间隙。此域的值对于批传送的端点及控制传送的端点无意义。对于同步传送的端点此域必需为 1，表示周期为 1ms。对于中断传送的端点此域值的范围为 1ms 到 255ms。

1.6 字符串描述符(String Descriptor)(0x03)

字符串描述符是一种可选的 USB 标准描述符，描述了如制商、设备名称或序列号等信息。如果一个设备无字符串描述符，则其它描述符中与字符串有关的索引值都必须为 0。字符串使用的是 Unicode 编码。

字符串描述符是用字符的形式描述设备、配置、接口、端点等信息。

字符串描述符以一种格式 2 类符值的方式存在：

1.> 显示语言的字符串描述符----------该字符串描述符表明了设备支持哪几种语言。

2.> 显示信息的字符串描述符----------用于描述具体的信息。

序号	域	大小	值	描述
0	bLength	1	数字	此表的字节数
1	bDecriptorType	1	常量	端点描述表类（此处应为 0x03）
2	wLANGID[0]	2	数字	语言标识（LANGID） 码 0
…	…	…	…	…
N	Strings	N	数字	语言标识（LANGID） 码 X

```
const u8 Virtual_Com_Port_StringLangID[VIRTUAL_COM_PORT_SIZ_STRING_LANGID] =       //
字符串描述
  {
    0x04,           //此描述表的字节数
    0x03,           //字串描述表类型（此处应为 0x03）
    0x09,0x04       //语言标识（LANGID） 码 0
                    //LangID = 0x0409: U.S. English
  };

const u8 Virtual_Com_Port_StringProduct[VIRTUAL_COM_PORT_SIZ_STRING_PRODUCT] =
  {
    38,         /* bLength */
    0x03,       /* bDescriptorType */
    /* Product name: "STM32 Virtual COM Port" */
    'S', 0, 'T', 0, 'M', 0, '3', 0, '2', 0, ' ', 0, 'V', 0, 'i', 0,
    'r', 0, 't', 0, 'u', 0, 'a', 0, 'l', 0, ' ', 0, 'C', 0, 'O', 0,
    'M', 0, ' ', 0, 'P', 0, 'o', 0, 'r', 0, 't', 0, ' ', 0, ' ', 0
  };
```

显示语言的字符串描述符与显示信息的字符串描述符的区别在于 Strings 项的不同，对于显示语言的字符串描述符来说 Strings 项由多个 wLANGID[n]数组元素组成，每个 wLANGID[n]是一个双字节的代表语言的 ID 值。而对于显示信息的字符串描述符而言，Strings 则是描述信息后的一组 UNICODE 编码。

为什么会出现这两种情况，原因在于访问字符串描述符的过程，主机请求访问某个字符串描述符的步骤分成两步：

第一步：获取语言信息----------首先主机向设备发送标准请求命令 Get_Descriptor，其参数为：描述符类型＝字符串描述符，字符串的索引值＝0，语言＝0，这样设备将返回显示语言的字符串描述符，从而主机知道了设备能支持哪些语言。

第二步：主机根据自己需要的语言，再次向设备发出标准请求命令 Get_Descriptor，其参数为：描述符类型=字符串描述符，字符串索引值=目标字符串索引值，语言=目标语言。这次设备将返回目标已经明确的显示信息的字符串描述符。

【说明 1：】只有字符串描述符的长度不是固定的，其长度为 N+2，其中 N 代表 Strings 项的字节数，2 代表字符串描述符的 bLength、bDescritorType 所占的两个字节。

1.7 IAD 描述符(Interface Association Descriptor)(0x0B)

USB 组合设备一般用 Interface Association Descriptor（IAD）实现，就是在要合并的接口前加上 IAD 描述符。

如果 USB 复合设备的固件中具有接口关联描述符 (IAD)，则 Windows 将枚举接口集合（就像每个集合都是一个设备），同时为每个接口集合分配一个物理设备对象 (PDO) 并将硬件与该 PDO 的兼容标识符 (ID) 关联。有关 IAD 的详细说明，请参阅 USB 接口关联描述符。本部分介绍分配给与 IAD 关联的接口集合的硬件 ID 和兼容标识符 (ID)。

```
typedef __packed struct USB_INTERFACE_ASSOCIATION_DESCRIPTOR
{
    u8    bLength;              /* bLength */              //描述符大小--0x08/8 个字节
    u8    bDescriptorType;      /* bDescriptorType */      //IAD 描述符类型
    u8    bFirstInterface;      /* bFirstInterface */      //起始接口
    u8    bInterfaceCount;      /* bInterfaceCount */      //接口数
    u8    bFunctionClass;       /* bFunctionClass */       //类型代码
    u8    bFunctionSubclass;    /* bFunctionSubclass */    //子类型代码
    u8    bFunctionProtocol;    /* bFunctionProtocol */    //协议代码
    u8    iFunction;            /* iFunction */            //描述字符串索引
};
```

1.8 设备类代码 bDeviceClass

接口类代码 bInterfaceClass，表示接口所属的类别，USB 协议根据功能将不同的接口划分成不的类，

其具体含义如下表所示：

值	类别	说明(类别)
0x00	设备	设备类型在接口描述符中说明
0x01	接口	音频类
0x02	都有	CDC 控制类
0x03	接口	人机接口类（HID）
0x05	接口	物理类
0x06	接口	图像类
0x07	接口	打印机类
0x08	接口	大数据存储类
0x09	设备	集线器类
0x0A	接口	CDC 数据类
0x0B	接口	智能卡类
0x0D	接口	安全类
0xDC	接口	诊断设备类
0xE0	接口	无线控制器类
0xEF	都有	混杂设备类
0xFE	接口	特定应用类（包括红外的桥接器等）
0xFF	都有	厂商定义的设备

1.9 描述符的编号及索引

1.8.1 一个 USB 设备只能拥有一个设备描述符，故设备描述符不需要编号。但设备描述符通常会提供设备最基本的文字描述信息，通常包含厂商、设备、产品的信息，故它拥有 3 个字符串描述符的索引，这 3 个索引将指向 3 个字符串描述，分别描述厂商信息、产品信息、设备序列号信息。简言之，设备描述符指示了设备有几种配置，及厂商、产品、设备序列号的字符串描述符索引。

1.8.2 配置描述符提供了相应的配置参数和查找参数：配置描述符编号 bCongfigurationValue、配置描述符的字符串描述符的索引。

1.8.3 接口描述提供了该接口的应用参数和查找参数：接口编号 bInterfaceNumber、接口描述符编号 bAlternateSetting、该接口描述符对应的字符串描述符的索引。

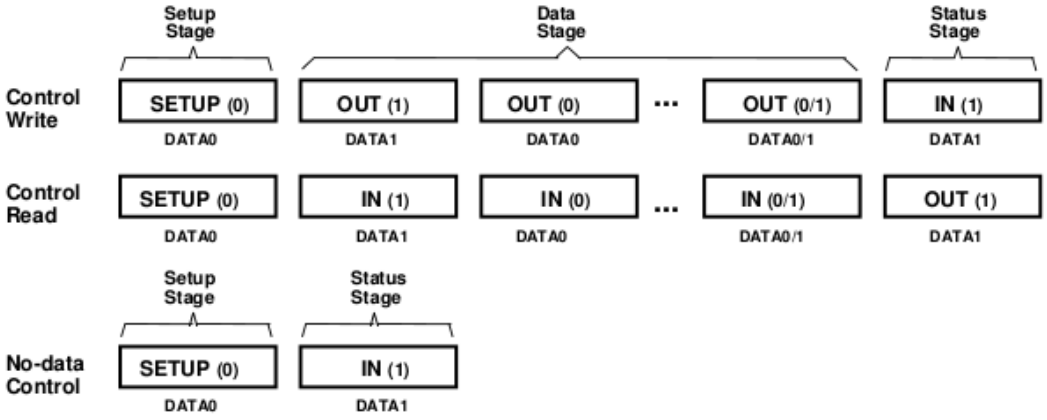
1.8.4 字符串描述符是对各描述符所需的字符信息描述的实现，每个描述符所需的字符信息描述的索引都将对应一个字符串描述符。但通常都不那么做，而是把所有的字符描述的实现都写在一个总的字符串描述符中，即字符串描述符的 bStrings 项，它们之间用索引来区分。

4. USB 的传输类型

USB 中有四种类型的端点，也就对应四种不同的传输方式，分别是控制传输、中断传输、同步传输和块传输。

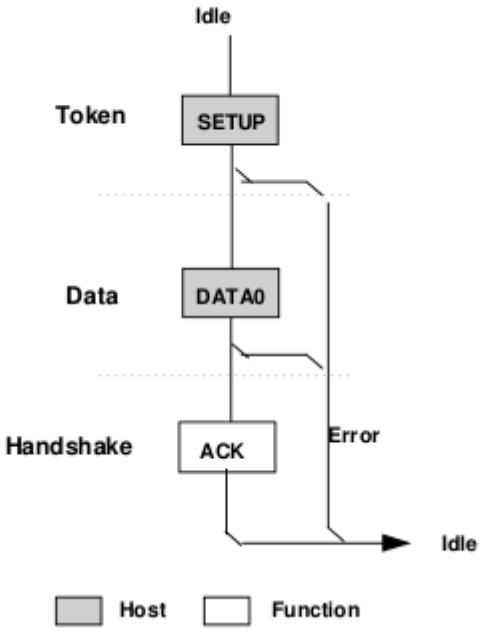
1.4 控制传输

1.4.1 控制传输的读写时序如下：



控制传输总共三个阶段，setup 阶段、数据阶段和状态阶段，其中数据阶段是可选的，而每个阶段都包含三个过程，即令牌过程、数据过程和握手过程。每个 USB 设备都必须具有控制传输功能，控制传输用于主机同设备的控制端点进行通信，通过读取设备的配置信息来完成对设备的枚举和配置。

1.4.2 setup 阶段



setup 阶段首先是 setup 令牌，然后是数据过程，最后是状态过程，对于数据过程只能使用 DATA0 包，设备在接收到 setup 数据包之后，需要返回 ACK 信号，如果接收数据错误，设备是不会返回握手包。setup 数据呢就是主机往设备发送的请求数据包，设备根据这个请求数据包来做相应的动作，例如：返回设备描述符或者直接进入状态阶段返回一个 0 长度的数据包。SETUP 传输呢有点类似于 OUT 传输，只不过 OUT 传输发送的是 OUT 令牌，SETUP 传输发送的是 SETUP 令牌。

1.4.3 数据阶段

如果是 OUT 传输，那么首先发送的是 OUT 令牌，如果 IN 传输，则发送的是 IN 令牌，然后是数据过程，数据过程必须以 DATA1 包开始，然后在 DATA0 和 DATA1 之间交替，注意数据过程的方向必须是同一个方向，即要么都是 IN 传输，要么都是 OUT 传输。

1.4.4 状态阶段

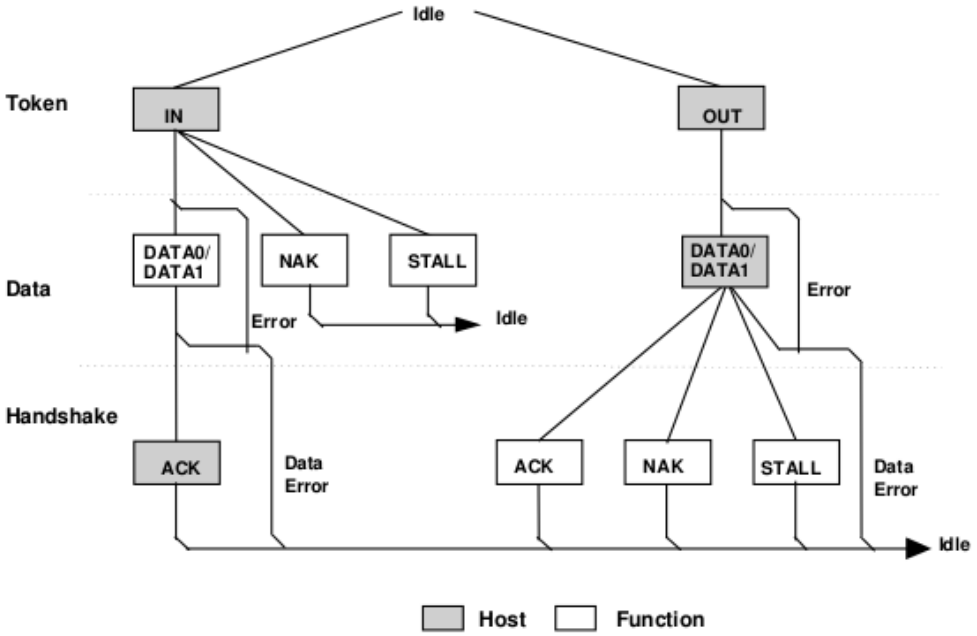
状态阶段的传输方向同数据阶段的传输方向刚好相反，即数据阶段是 IN 传输，状态阶段就是 OUT 传输，数据阶段是 OUT 传输，状态阶段就是 IN 传输。如果没有数据阶段，那就是只能是 IN 传输。状态阶段的响应信息如图所示：

Status Response	**Control Write Transfer** (sent during data phase)	**Control Read Transfer** (sent during handshake phase)
Function completes	Zero-length data packet	ACK handshake
Function has an error	STALL handshake	STALL handshake
Function is busy	NAK handshake	NAK handshake

状态阶段的数据过程使用的 DATA1 包，如果是控制写，设备在正确收到数据包之后将返回一个 0 长度的数据包。注意这个 0 长度数据和没有数据概念是不一样的，0 长度数据有数据的包头，只是后面没有数据罢了。对于控制读，主机在接收数据之后，将返回 ACK 握手信息。

1.5 中断传输

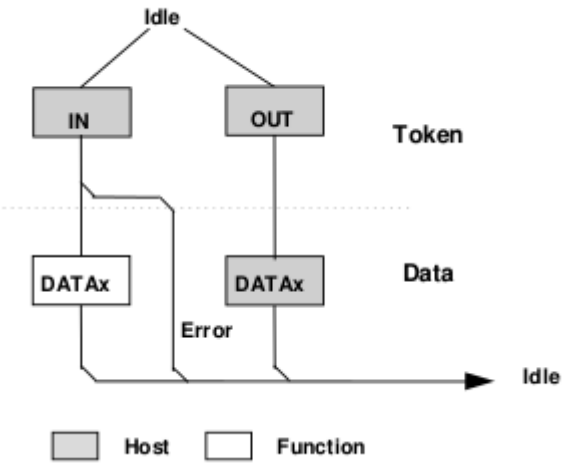
中断传输呢分为 IN 和 OUT 传输，如果是 IN 传输，设备返回数据或者 NAK、STALL 握手信息。如果端点没有新的中断信息返回，在数据过程中设备返回的是 NAK 握手信息，如果此时端点已经被设置为暂停了，设备返回的是 STALL 握手信息，如果设备返回的是中断信息数据包，主机必须返回一个 ACK 握手信息给设备，如果数据数据接收错误，将不会返回握手信息。IN、OUT 传输过程如图所示：



中断传输一般用于这种具有固定速率、数据量少的数据传输，例如 USB 鼠标、键盘就是采用的中断传输。

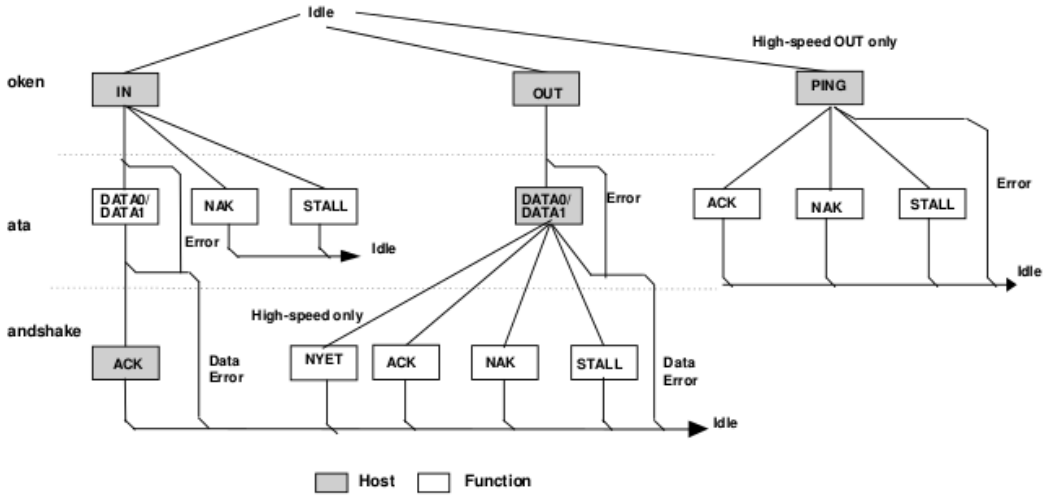
1.6 同步传输

同步传输也叫做等时传输，也分为 IN 和 OUT 传输，但是注意没有握手过程，所以说它并不保证数据传输是否正确性，但是要保证数据传输的实时性，所以这种传输方式一般用于音频和视频流的数据传输，例如你的 USB 摄像头就是采用的这种传输方式，传输过程如下：



1.7 块传输

块传输也叫做批量传输，块传输并不追求数据传输的时间，但是必须保证数据传输的正确性，例如 U 盘就是采用的这种传输方式，传输过程如下：



为了保证数据传输的正确性，USB 采用的是错误检测机制和重试机制来确保数据传输正确，当然它也分三个过程，令牌过程、数据过程和握手过程，其中 PING 令牌和 NYET 令牌只用于高速设备。

5. 端点寄存器

EndPoint	Direction	Register	Length	PMA_Addr_Begin	PMA_Addr_End
40006000			Hex	Hex	Hex
EP0	Tx0	AddrTx0	4	40006000	40006003
		CountTx0	4	40006004	40006007
	Rx0	AddrRx0	4	40006008	4000600B
		CountRx0	4	4000600C	4000600F
EP1	Tx1	AddrTx1	4	40006010	40006013
		CountTx1	4	40006014	40006017
	Rx1	AddrRx1	4	40006018	4000601B
		CountRx1	4	4000601C	4000601F
EP2	Tx2	AddrTx2	4	40006020	40006023
		CountTx2	4	40006024	40006027
	Rx2	AddrRx2	4	40006028	4000602B
		CountRx2	4	4000602C	4000602F
EP3	Tx3	AddrTx3	4	40006030	40006033
		CountTx3	4	40006034	40006037
	Rx3	AddrRx3	4	40006038	4000603B
		CountRx3	4	4000603C	4000603F
EP4	Tx4	AddrTx4	4	40006040	40006043
		CountTx4	4	40006044	40006047
	Rx4	AddrRx4	4	40006048	4000604B
		CountRx4	4	4000604C	4000604F
EP5	Tx5	AddrTx5	4	40006050	40006053
		CountTx5	4	40006054	40006057
	Rx5	AddrRx5	4	40006058	4000605B
		CountRx5	4	4000605C	4000605F
EP6	Tx6	AddrTx6	4	40006060	40006063
		CountTx6	4	40006064	40006067
	Rx6	AddrRx6	4	40006068	4000606B
		CountRx6	4	4000606C	4000606F
EP7	Tx7	AddrTx7	4	40006070	40006073
		CountTx7	4	40006074	40006077
	Rx7	AddrRx7	4	40006078	4000607B
		CountRx7	4	4000607C	4000607F
EP8	Tx8	AddrTx8	4	40006080	40006083
		CountTx8	4	40006084	40006087
	Rx8	AddrRx8	4	40006088	4000608B
		CountRx8	4	4000608C	4000608F

6. 实现一个 USB 设备的步骤

1. 根据应用选择合适的 USB 类实现
2. 根据所选的 USB 类协议，完成各个描述符（包括设备描述符、配置描述符、接口描述符和字符串描述符）
3. 根据描述符，初始化端点数目，分配各端点所需使用的 packet buffer
4. 初始化所使用的端点，配置端点的传输类型、方向、packet buffer 地址和初始状态
5. 在需要发送或接收数据的时候，使能端点
6. 在该端点的中断回调函数中，处理数据，如果需要则使能一下次传输

1.1 端点：

1.1.1 根据应用定义需要使用的端点数

Usb_conf.h

```
#define EP_NUM (4)              //定义端点数量
```

1.1.2 初始化端点

Usb_prop.c

```
SetEPType(ENDP1, EP_BULK);              //设置端点 1 为进批量传输
SetEPTxAddr(ENDP1, ENDP1_TXADDR);       //设置端点发送地址
SetEPTxStatus(ENDP1, EP_TX_NAK);        //设置端点 1 的发送不响应
SetEPRxStatus(ENDP1, EP_RX_DIS);        //设置端点 1 不接收
```

1.1.3 使能需要的端点

对 IN 端点：

```
  SetEPTxCount(ENDP1, Count);//设置端点长度
   SetEPTxValid(ENDP1);                 //使能端点 1
```

对 OUT 端点：

```
   SetEPRxValid(ENDP3);                 //使能端点 3
```

1.1.4 端点处理函数

四、 STM32 USB 库

1. USB 核心库

：该层管理使用 USB IP 硬件和 USB 标准协议的直接传输。USB 库内核遵从 USB2.0 标准并和标准的 STM32F10xxx 固件库分离。

USB 库内核模块：

Usb_type.h---------库内核用到的数据类型，本文件用于保证 USB 库的独立性。

Usb_reg (.h,.c)----硬件抽象层

Usb_int.c----------正确传输中断服务程序

Usb_init (.h,.c)---USB 初始化

Usb_core (.h,.c)---USB 协议管理（服从 USB2.0 规范的第九章）

Usb_mem (.h,.c)----数据传输管理（从包存储器区域发出的或者发往包存储器区域的）

Usb_def.h----------USB 定义

1.1 usb_type.h：

该文件提供了库中使用的主要数据类型，这些数据类型和使用的微控制器家族有关。

注意：USB 库中类型的定义和 STM32F10xxx 固件库中相同， 这保证了整个代码的一致性。

1.2 Usb_reg (.c,.h)：

Usb_reg 模块实现了硬件抽象层，它提供了访问 USB 宏单元寄存器的一组基本函数。

注意：可用函数有两种调用方式：

— 作为宏：_调用方式是：函数名（参数 1，2…）

— 作为子程序：调用方式是：函数名（参数 1，2…）

1.3 usb_int (.c , .h)：

usb_int 模块处理正确的传输中断服务程序，提供了 USB 协议事件和库内核之间的连接。

The STM32F10xxx USB 外设提供两种正确传输例程：

1）低优先级中断：由 CTR_LP()函数管理，在控制，中断，批量模式下使用（单缓存模式）。

2）高优先级中断：由 CTR_HP()函数管理，在快速传输方式（如同步，批量模式）（双缓存模式）

1.4 usb_core (.c , .h)：

usb_core 模块是库的内核，实现了 USB 2.0 规范中第九章描述的所有功能。

可用的子例程覆盖了和控制端点（EP0）有关的 USB 标准请求处理以及为完成列举过程所需代码的提供。

为了处理设置事务的不同阶段，内核实现了一个状态机。

USB 内核模块还利用结构体 User_Standard_Requests 实现了标准请求和用户实现之间的动态接口。

只要需要，USB 内核可以将一些类专用的请求和总线事件分配给用户程序处理。用户处理过程在

Device_Property 结构中给出。

2. USB 应用库

应用接口模块是作为一个模版提供给用户的，开发人员可以根据实际的应用对模版进行裁剪，
应用接口模块：
usb_istr (.c,.h)--------USB 中断处理函数
usb_conf.h----------USB 配置文件
usb_prop (.c, .h) ----USB 应用相关属性，用于上层协议处理，比如 HID 协议，大容量存储设备协议
usb_endp.c----------CTR 中断处理程序（非控制端点）
usb_pwr (.h, .c) -----USB 电源管理模块
usb_desc (.c, .h) ----USB 描述符

1.1 usb_istr(.c)：

USB_istr 提供了一个名为 USB_Istr() 的函数，该函数用于处理所有的 USB 宏单元中断。
每一个 USB 中断源有一个名为 XXX_Callback（如 RESET_Callback）的回调函数，这些回调函数用
于实现用户中断处理器。为了启用回调程序，必须在 USB 配置文件 USB_conf.h 中
定义名为 XXX_Callback 的与处理程序开关。

1.2 usb_conf(.h)：

usb_conf.h 用于：
1）定义 BTABLE 和 PMA 中的所有端点地址
2）定义相应事件中的中断掩码

1.3 usb_endp (.c):

USB_endp 模块处理除端点 0（EP0）外所有的 CTR 的正确传输程序。
为了启用回调处理器进程，必须在 USB_conf.h 中定义一个名为 EPx_IN_Callback (IN 传输) 和 EPx_OUT_Callback (OUT 传输)的预处理器

1.4 usb_prop (.c , .h):

USB_prop 模块实现了 USB 内核使用的 Device_Property, Device_Table 和 USER_STANDARD_REQUEST 结构。
设备属性的实现:实现相关设备的 USB 协议，例如初始化、SETUP 包、IN 包、OUT 包等等

1.5 usb_pwr (.c , .h):

本模块管理 USB 设备的电源，包含处理上电、掉电、挂起和恢复事件

五、 USB 的请求

1、 usb 请求包结构体组成（USB_SETUP_PACKET）

bmRequestType + bRequest + wValue + wIndex + wLength

在 USB 通讯里，从主控器发出来的第一个配置包就是设备描述符配置包，目的只有一个，就是获取插入的 USB 属性，以便加载合适的驱动程序。现在就来详细地分析一下设备描述符包的定义。

在 USB2.0 的协议里找到 9.3 USB Device Requests 里就找到这个结构的定义，这里我使用 C 的定义结构如下：

常规形式：

```
typedef struct _USB_SETUP_PACKET
{
    REQUEST_TYPE      bmRequestType;
    BYTE        bRequest;
    WORD_BYTE    wValue;
    WORD_BYTE    wIndex;
    WORD        wLength;
} USB_SETUP_PACKET;
```

另一种定义如下（联合）：

```
typedef __packed struct _USB_SETUP_PACKET
{
    REQUEST_TYPE   bmRequestType;
    U8   bRequest;
    __packed union
    {
        U16 wValue;
        __packed struct
        {
            U8   wValueL;
            U8   wValueH;
        };
    };
    __packed union
    {
        U16 wIndex;
        __packed struct
        {
            U8   wIndexL;
            U8   wIndexH;
        };
```

```
    };
    U16 wLength;
} USB_SETUP_PACKET;
```

bmRequestType 是包含有下面几方面的内容：

```
typedef __packed struct _REQUEST_TYPE {
    U8      Recipient : 5;
    U8      Type      : 2;
    U8      Dir       : 1;
} REQUEST_TYPE;
```

2、 请求包说明

分析（举例）数据包(低位在前)：

80 06 00 01 00 00 40 00

常规形式

序号	域	字节	值	描述
0	bmRequestType	1	0x80	请求目标/方向
1	bRequest	1	0x06	请求类型
2	wValue	2	0x0001	传送当前请求的参数，随请求不同而变
3	wIndex	2	0x0000	当 bmRequestType 的 Recipient 字段为接口或端点时，wIndex 域用来表明是哪一个接口或端结。
4	wLength	2	0x0040	第二阶段数据传输的最大长度

联合形式

序号	域		字节	值	描述
0	bmRequestType		1	0x80	请求目标/方向
1	bRequest		1	0x06	请求类型
2	wValue	wValueL	1	0x01	表示描述符号的索引
		wValueH	1	0x00	标识描述表类型(Descriptor Types)
3	wIndex	wIndexL	1	0x00	当bmRequestType的Recipient字段为接口或端点时，wIndex 域用来表明是哪一个接口或端结。
		wIndexH	1	0x00	
4	wLength		2	0x0040	第二阶段数据传输的最大长度

1.1 bmRequestType

在这一个字节里，又按位分为：

D7 D6 D5 D4 D3 D2 D1 D0

D7 位(Dir)是表示后面传送数据的方向位。0＝主机至设备，1＝设备至主机

在这里，收到的数据是 80，就表示从 USB 设备里发送数据给 PC。

这里的 80，就是 D7 位为 1。

D6-D5 位(Type)是请求主分类型

0 是表示标准的请求（REQUEST_STANDARD）。

1 是表示类别的请求（REQUEST_CLASS）。

2 是表示厂商的请求。

3 是保留。

D4-D0 位（**Recipient**）是表求接收这个包的接口。

0 是表示 USB 设备接收（主机到设备）。

1 是表示接口接收。

2 是表示端点接收。

3 是表示其它接收，不知道的。

4-31 是保留。

1.2 bRequest（标准请求类型）

bRequest 是本描述符的请求类型，也就是后面发送的数据是什么样的东西。由于 USB 里有很多配置信息，比如获取设备描述符，又有设置 USB 地址等等，就是通过这个字节来区分的。

从 USB 协议里查找表 9-4，就可看到如下的编码：

编号	值	名称	说明
0	0x00	GET_STATUS	用来返回特定接收者的状态
1	0x01	CLEAR_FEATURE	用来清除或禁止接收者的某些特性
2	0x02	保留	为将来保留
3	0x03	SET_FEATURE	用来启用或激活命令接收者的某些特性
4	0x04	保留	为将来保留
5	0x05	SET_ADDRESS	用来给设备分配地址
6	0x06	GET_DESCRIPTOR	用于主机获取设备的特定描述符
7	0x07	SET_DESCRIPTOR	修改设备中有关的描述符，或者增加新的描述符
8	0x08	GET_CONFIGURATION	用于主机获取设备当前设备的配置值(注同上面的不同)
9	0x09	SET_CONFIGURATION	用于主机指示设备采用的要求的配置
10	0x0A	GET_INTERFACE	用于获取当前某个接口描述符编号
11	0x0B	SET_INTERFACE	用于主机要求设备用某个描述符来描述接口
12	0x0C	SYNCH_FRAME	用于设备设置和报告一个端点的同步帧

3、 请求分类举例说明

六、 USB 的描述符与命令请求

1、 描述符

1.1 概述

所谓描述符，就是用于描述设备特性的具有特定格式排列的一种数据组织结构。

1.2 作用

描述符的作用在于设备向主机汇报自己的信息、特征，主机根据这些信息从而加载相应

1.3 分类

描述符分为三大类：标准描述符、设备类描述符、厂商描述符。
除字符串描述符可选外，任何设备都必须包含剩下的几种标准描述符。
在 USB1.0 中规定了 5 种标准的描述符：设备描述符
配置描述符
接口描述符
端点描述符
字符串描述符
规定的设备类描述符有：集线器类描述符、人机接口类描述符。

1.1 描述符类型说明 bDescriptorType

其中 bDescriptorType 为描述符的类型，其含义可查下表
(此表也适用于标准命令 Get_Descriptor 中 wValue 域高字节的取值含义)：

1.2 描述符的编号及索引：

1.2.1、一个 USB 设备只能拥有一个设备描述符，故设备描述符不需要编号。但设备描述符通常会提供设备最基本的文字描述信息，通常包含厂商、设备、产品的信息，故它拥有 3 个字符串描述符的索引，这 3 个索引将指向 3 个字符串描述，分别描述厂商信息、产品信息、设备序列号信息。简言之，设备描述符指示了设备有几种配置，及厂商、产品、设备序列号的字符串描述符索引。
1.2.2、配置描述符提供了相应的配置参数和查找参数：配置描述符编号 bCongfigurationValue、配置描述符的字符串描述符的索引。

1.2.3、接口描述提供了该接口的应用参数和查找参数：接口编号 bInterfaceNumber、接口描述符编号 bAlternateSetting、该接口描述符对应的字符串描述符的索引。
1.2.4、字符串描述符是对各描述符所需的字符信息描述的实现，每个描述符所需的字符信息描述的索引都将对应一个字符串描述符。但通常都不那么做，而是把所有的字符描述的实现都写在一个总的字符串描述符中，即字符串描述符的 bStrings 项，它们之间用索引来区分。

2、 HID 相关描述符

USB 设备中有一大类就是 HID 设备，即 Human Interface Devices，人机接口设备。这类设备包括鼠标、键盘等，主要用于人与计算机进行交互。它是 USB 协议最早支持的一种设备类。HID 设备可以作为低速、 全速、高速设备用。由于 HID 设备要求用户输入能得到及时响应，故其传输方式通常采用中断方式。

在 USB 协议中，HID 设备的定义放置在接口描述符中，USB 的设备描述符和配置描述符中不包含 HID 设备的信息。因此，对于某些特定的 HID 设备，可以定义多个接口，只有其中一个接口为 HID 设备类即可。

当定义一个设备为 HID 设备时，其设备描述符(DeviceDescriptor)应为：

bDeviceClass=0

bDeviceSubClass=0

bDeviceProtocol=0

其接口描述符(INTERFACE_DESCRIPTOR)应该

bInterfaceClass=0x03

另外(接口描述符)：

对无引导的 HID 设备，子类代码 bInterfaceSubClass 应置 0，此时 bInterfaceProtocol 无效，置零即可。即为：

bInterfaceClass=0x03

bInterfaceSubClass=0

bInterfaceProtocol=0

对支持引导的 USB 设备，子类代码 bInterfaceSubClass 应置 1，此时 bInterfaceProtocol 可以为 1 或 2，1 表示键盘接口，2 表示鼠标接口。其参考设置如下：

bInterfaceClass=0x03

bInterfaceSubClass=1

bInterfaceProtocol=1 或 2

HID 设备支持 USB 标准描述符中的五个：设备描述符、配置描述符、接口描述符、端点描述符、字符串描述符。除此之外，HID 设备还有三种特殊的描 述符：HID 描述符、报告描述符、物理描述符。一个 USB 设备只能支持一个 HID 描述符，但可以支持多个报告描述符，而物理描述符则可以有也可以没有。

1.1、 HID 描述符(0x21)

HID 描述符用于识别 HID 设备中所包含的额外描述符，例如报告描述符或物理描述符等。其格式如下：

HID 描述符格式：

序号	域	大小	值	描述
0	bLength	1	数字	HID 描述符长度
1	bDecriptorType	1	常量	HID 描述符类型，值为 0x21
2	bcdHID	2	BCD	HID 设备所遵循的 HID 版本号，为 4 位 16 进制的 BCD 码数据。1.0 即 0x0100，1.1 即 0x0101，2.0 即 0x0200。
3	bCountryCode	1		HID 设备国家/地区代码。
4	bNumDescriptor	1	数字	HID 设备支持的其他设备描述符的数量。由于 HID 设备至少需要包括一个报告描述符，故其值至小为 0x01。
5	bDescriptorType	1	常量	HID 描述符附属的类别描述符长度。
6	wDescriptorLength	2	数字	可选字段，用于表示 HID 描述符附属的类别描述符类型及长度。
7	bDescriptorType	1	常量	HID 描述符附属的类别描述符长度。
8	wDescriptorLength	2	数字	可选字段，用于表示 HID 描述符附属的类别描述符类型及长度。

【说明 1:】HID 设备类描述符并不是说仅用这一个描述符就可描述清楚这类设备，而是指 HID 设备除包含所有的标准描述符外，还需这个 HID 设备来补充描述。也就是说，在使用一般的设备时，只需使用标准的描述符就可描述清楚，而若使用 HID 设备时，除了要使用全部的标准的描述符外还需 HID 描述符来补充描述。同时，从 HID 描述符中看出，它还将引出 HID 的报告描述符，在此不讲述。可以这么说，设备类描述符是作为一个对标准描述进行补充描述的描述符。

1.2、 HID 报告描述符(0x22)

USB HID 设备是通过报告来给传送数据的，报告有输入报告和输出报告。输入报告是 USB 设备发送给主机的，例如 USB 鼠标将鼠标移动和鼠标点击等信息返回给电脑，键盘将按键数据数据返回给电脑等；输出报告是主机发送给 USB 设备的，例如键盘上的数字键盘锁定灯和大写字母锁定灯等。报告是一个数据包，里面包含的是所要传送的数据。输入报告是通过中断输入端点输入的，而输出报告有点区别，当没有中断输出端点时，可以通过控制输出端点 0 发送，当有中断输出端点时，通过中断输出端点发出。
而报告描述符，是描述一个报告以及报告里面的数据是用来干什么用的。通过它，USB HOST 可以分析出报告里面的数据所表示的意思。它通过控制输入端点 0 返回，主机使用获取报告描述符命令来获取报告描述符，注意这个请求是发送到接口的，而不是到设备。一个报告描述符可以描述多个报告，不同的报告通过报告 ID 来识别，报告 ID 在报告最前面，即第一个字节。当报告描述符中没有规定报告 ID 时，报告中就没有 ID 字段，开始就是数据

报告描述符主要是为了描述报告的结构、用途。
指定位域作用时，同时指明用途页、用途。
指明用途可以一个个指定、可以指明最大值和最小值。
Sel 和 Ary：这个在实例中阐明。
基本用途类型：集合、控制、数据。
鼠标、游戏杆、键盘都属于集合类用途。
音量控制、键盘 LED 开关控制、鼠标按键控制属于控制类用途。
选择器、动态标志、动态值属于数据类用途。

关键字：
Usage Page
Logical Minimum
Logical Maximum
Report Size
Report ID
Report Count

Collection
Collection Items
Global Items
Local Items

Generic Desktop

Collection

1.3、 HID 报告描述符格式

这里定义了一个输入和输出 64 字节数据的报告描述符。

```
const unsigned char HID_ReportDescriptor [] =
{
0x06,0xA0,0xFF,//用法页(FFA0h, vendor defined)
0x09, 0x01,//用法(vendor defined)
0xA1, 0x01,//集合(Application)
0x09, 0x02 ,//用法(vendor defined)
0xA1, 0x00,//集合(Physical)
0x06,0xA1,0xFF,//用法页(vendor defined)
//输入报告
0x09, 0x03 ,//用法(vendor defined)
0x09, 0x04,//用法(vendor defined)
0x15, 0x80,//逻辑最小值(0x80 or -128)
0x25, 0x7F,//逻辑最大值(0x7F or 127)
0x35, 0x00,//物理最小值(0)
0x45,0xFF,//物理最大值(255)
0x75, 0x08,//报告长度 Report size (8 位)
0x95, 0x40,//报告数值(64 fields)
0x81, 0x02,//输入(data, variable, absolute)
//输出报告
0x09, 0x05,//用法(vendor defined)
0x09, 0x06,//用法(vendor defined)
0x15, 0x80,//逻辑最小值(0x80 or -128)
0x25, 0x7F,//逻辑最大值(0x7F or 127)
0x35, 0x00,//物理最小值(0)
0x45,0xFF,//物理最大值(255)
0x75,0x08,//报告长度(8 位)
0x95, 0x40,//报告数值(64 fields)
0x91, 0x02,//输出(data, variable, absolute)
0xC0,//集合结束(Physical)
0xC0//集合结束(Application)
};
```

这样，后面数据的输入和输出都必须满足报告的格式才能够进行传输。

1.4、 HID 物理描述符(0x23)

HID 设备的物理描述符主要用于报告物理设备的激活信息，其类型值为 0x23,它是可选的，对大部分设备不需要使用此描述符。

1.5、 HID 库

1.4.1 HID 请求代码（HID Request Codes）

请求	值	描述
Get_Report	*0x01*	主机用控制传输从设备接收数据，所有 HID 类设备都要支持这个请求
Get_Idle	*0x02*	主机读取设备当前的空闲速率，设备可以不支持此请求
Get_Protocol	*0x03*	主机获得设备的当前活动是引导协议还是报告协议
Set_Report	*0x09*	设备用控制传输接收主机的数据，设备可以不支持此请求
Set_Idle	*0x0A*	设置闲置状态，设备可不支持此请求
Set_Protocol	*0x0B*	在引导协议和报告协议间切换，设备如果支持系统引导（如键盘和鼠标），就必须支持 Get_Protocol 和 Set_Protocol 请求

1.4.2 HID 报告类型（HID Report Types）

请求	值	功能描述
Report_Input	*0x01*	输入
Report_Output	*0x02*	输出
Report_Feature	*0x03*	特性

1.4.3 HID 用途页（Usage Pages）

请求	值	描述
Page_Undefined	*0x00*	
Page_Generic	*0x01*	
Page_Simulation	*0x02*	
Page_VR	*0x03*	
Page_Sport	*0x04*	
Page_Game	0x05	
Page_Dev_Controls	*0x06*	
Page_Keyboard	*0x07*	
Page_Led	*0x08*	
Page_Button	*0x09*	
Page_Ordinal	*0x0A*	
Page_Telephony	*0x0B*	
Page_Consumer	*0x0C*	
Page_Digitizer	*0x0D*	
Page_Unicode	*0x10*	

Page_Alphanumeric	*0x14*	

1.4.4 通用桌面用途（Generic Desktop Page-0x01）

请求	值	描述
Generic_Pointer	*0x01*	
Generic_Mouse	*0x02*	
Generic_Joystick	*0x04*	
Generic_Gamepad	*0x05*	
Generic_Keyboard	*0x06*	
Generic_Keypad	*0x07*	
Generic_X	*0x30*	
Generic_Y	*0x31*	
Generic_Z	*0x32*	
Generic_RX	*0x33*	
Generic_RY	*0x34*	
Generic_RZ	*0x35*	
Generic_Slider	*0x36*	
Generic_Dial	*0x37*	
Generic_Wheel	*0x38*	
Generic_HatWitch	*0x39*	
Generic_Counted_Buffer	*0x3A*	
Generic_Byte_Count	*0x3B*	
Generic_Motion_Wakeup	*0x3C*	
Generic_VX	*0x40*	
Generic_VY	*0x41*	
Generic_VZ	*0x42*	
Generic_VBRX	*0x43*	
Generic_VBRY	*0x44*	
Generic_VBRZ	*0x45*	
Generic_VNO	*0x46*	
Generic_System_Ctl	*0x80*	
Generic_Sysctl_Power	*0x81*	
Generic_Sysctl_Sleep	*0x82*	
Generic_Sysctl_Wake	*0x83*	
Generic_Sysctl_Context_Menu	*0x84*	
Generic_Sysctl_Main_Menu	*0x85*	
Generic_Sysctl_ App_Menu	*0x86*	
Generic_Sysctl_Help_Menu	*0x87*	
Generic_Sysctl_Menu_Exit	*0x88*	
Generic_Sysctl_Menu_Select	*0x89*	
Generic_Sysctl_Menu_Right	*0x8A*	

Generic_Sysctl_Menu_Left	*0x8B*	
Generic_Sysctl_Menu_Up	*0x8C*	
Generic_Sysctl_Menu_Down	*0x8D*	

1.4.5 区域代码（bCountryCode）

CODE (decimal)	**Country**	CODE (decimal)	**Country**
00	Not Supported	18	Netherlands/Dutch
01	Arabic	19	Norwegian
02	Belgian	20	Persian (Farsi)
03	Canadian-Bilingual	21	Poland
04	Canadian-French	22	Portuguese
05	Czech Republic	23	Russia
06	Danish	24	Slovakia
07	Finnish	25	Spanish
08	French	26	Swedish
09	German	27	Swiss/French
10	Greek	28	Swiss/German
11	Hebrew	29	Switzerland
12	Hungary	30	Taiwan
13	International (ISO)	31	Turkish-Q
14	Italian	32	UK
15	Japan (Katakana)	33	US
16	Korean	34	Yugoslavia
17	Latin American	35	Turkish-F
		36-255	Reserved

3、 HUB 描述符(0x29)

10、描述符的获取

9.1、获取描述符的命令格式

命令码 CmdCode = GetDescriptor，格式如下：

bmRequestType	bRequest	wValue	wIndex	wLength
0x80	0x60	类型和索引	0 或语言 ID	描述符长度

bmRequestType---

bRequest ----------

wValue----------其高字节 wValue_H 指明要获取的描述符类型（实际只有 3 种类型：设备描述符类型、配置描述符类型、字符串描述符类型），低字节 wValue_L 指明目标描述符的索引，然而 wValue_L 的值只对配置描述符和字符串描述符有效，而对设备描述符无效。

wIndex----------只对字符串描述符有意义，对其它描述符时该值为 0.。当然对于字符串描述符时，其值也可为 0，表示要获取“显示语言的字符串描述符” ，若为其它值则代表了确定的语言 ID，即表明要获取指定了语言的“显示信息的字符串描述符” 。

wLength--------主机要求的返回的描述符长度。如果 wLength 大于实际的描述符长度，则以实际描述符长度为准；如果 wLength 小于实际描述符长度，则以 wLength 值为准。

9.2、获取描述符的过程

获取描述符属于枚举的过程，其整个过程当然必经 Setup 传输的 3 大过程：Setup 过程、数据过程、状态信息过程。

首先，在 Setup 过程中，主机发送 GetDescriptor 命令。若成功，设备就开始准备数据，通信将继续向前推进，进入数据过程。

然后，在数据过程中，主机启动 IN 事件，设备就把准备好的数据（描述符）发送出去。若成功，则通信继续向前推进，进入状态信息过程。

最后，在状态信息过程，主机发送通信过程的信息状态，祝贺并告知通信完美结束。

9.3、例程说明

9.3.1、获取配置描述符

对于主机来说，配置是广义的，包括狭义的配置、接口配置、端点配置等，而接口配置、端点配置等
都隶属于标准配置描述符，故主机若要求获取配置描述符时，实际上是要求获取除设备描述符和字符串描述符以外的所有描述符。

对于只有标准描述符的设备而言，当主机要求或者配置描述符时，需设备按照顺序把标准配置描述符、标准接口描述符、标准端点描述符一次性发给主机。

所以，通常在写程序时，会将广义上的“配置”打成一个包，在包中，由标准配置描述符引领，按照发送顺序依次实现标准接口描述符、标准端点描述符等。这样做的理由是，在标准配置描述符中有一项 wTotalLength，它代表广义上的配置包描述符总长度，根据这个参数就可把广义的配置包描述符一起发给主机，以避免多个描述符时的多次传输。

bmRequestType	bRequest	wValue	wIndex	wLength
0x80	0x60	类型和索引	0 或语言 ID	描述符长度

wValue _H = 配置描述符类型。
wValue _L = 配置描述符编号(索引)，实际为 bCongfigurationValue 值。
wIndex = 0 。
wLength，其值由主机自己规定。
因为，是按确定的顺序发送的，故主机解析的结果也将一一对应。
下面是一个广义配置包描述符的结构模板：

```
uint8_t USB_ConfigDescriptor[] = {
        标准配置描述符的实现；
        标准接口描述符的实现；
标准设备类描述符的实现；
标准端点描述符的实现；
}
```

9.3.2、获取配置描述符

从设备描述符到端点描述符，需要许多的信息描述，即需要许多字符串描述符来描述它们的信息。然而，标准字符串中没有总长度显示项 wTotalLength，且每个字符串描述符的格式都一样，所以不可能向获取配置描述符那样，用广义的配置包描述符一起发给主机，况且有些字符串描述符不是必须的，所以很难做到统一的格式。

不过，为了方便管理，在编程时通常还是把所有的字符串描述符组织在一起，不过主机在访问它们时只能一个一个的访问，而不能打包访问，它们之间的选取是依赖各个字符串描述符的长度进行跳过操作来实现的。所以这种组织在一起，只是为了方便管理或好看，而没有其它任何作用，组织的形式通常以“显示语言的字符串描述符”领头，模板如下：

```
uint8_t USB_StringDescriptor[] = {
        显示语言的标准字符串描述符;
        显示信息的标准字符串描述符 1 ;
            ... ...
        显示信息的标准字符串描述符 n ;
    };
```

在字符串描述符组织中，各个字符串是怎么区分的？是应用程序，因为这个组织是编写应用程序时自己规定内部秩序的，故组织中各个字符串描述符对应的索引，程序员当然知道。

bmRequestType	bRequest	wValue	wIndex	wLength
0x80	0x60	类型和索引	0 或语言 ID	描述符长度

wValue _H = 字符串描述符类型。

wValue _L = 字符串描述符对应的编号(索引)。

wIndex = 0 或 ID。

wLength，其值由主机自己规定。

在获取字符串描述符的第一步：获取设备所支持的语言中，wValue _L = 0，wIndex = 0 ，设备将把显示语言的标准字符串描述符发给主机，主机会从中挑选一种语言。

在获取字符串描述符的第二步：获取显示信息的字符串描述符中，wValue _L = 目标字符串对应的编号，wIndex=语言 ID，设备则把确定的字符串描述符发给主机。

七、 USB 设备请求

USB 设备请求命令 ： bmRequestType + bRequest + wValue + wIndex + wLength

标准请求代码:

请求	值	功能描述
GetStatus	*0x00*	*读取 USB 设备、接口或端点状态*
ClearFeature	*0x01*	*清楚或禁止 USB 设备、接口或端点的某些特性*
SetFeature	*0x03*	*设置或使能 USB 设备、接口或端点的某些特性*
SetAddress	*0x05*	*分配设备地址*
GetDescriptor	*0x06*	*读取设备描述符*
GetDescriptor	*0x07*	*更新已有的描述符或添加新的描述符*
GetConfiguration	*0x08*	*读取当前的配置值*
SetConfiguration	*0x09*	*为 USB 设备选择一个合适的配置*
GetInterface	*0x0A*	*读取 USB 指定接口的当前可替换设置值*
SetInterface	*0x0B*	*为 USB 指定接口选择一个合适的可替换设置*
SynchFrame	*0x0C*	*读取 USB 同步端点所指定的帧序号*

八、 USB 复合设备

九、 USB 组合设备

十、 USB 大容量存储设备（MSC）

关键词

1、 SCSI 协议

1.1、SCSI 指令概述

SCSI（small computer system interface）是小型计算机系统的缩写，有一套完整的协议规定其命令和命令数据的响应。

1） Host 和 Device 间数据通讯协议是 Bulk-only Transport。也称为 BBB 协议，这是与 CBI 对应的一种说法。因为 CBI 是指 Commond、Bulk 和 Interrupt。而对于 Bulk-only 所有的传输都是通过 BULK EP 完成。
2） USB Device 内部和数据存储介质之间通信协议为 SCSI（Small Computer System Interface）。
3） scsi 的命令有很多，但 u 盘中常用的就几个：INQUIRY，READ CAPACITY , READ(10),WRITE(10)等命令。

1.2、U 盘需要处理的命令

1） **inquiry**：设备的一个描述，告诉 host 你的设备是什么，名字叫什么，用的什么协议，这里用的 SCSI 协议—SPC2

2） **READ FORMAT CAPACITIES**：读格式容量
（The READ FORMAT CAPACITIES command allows the host to request a list of the possible capacities that can be formatted on the currently installed medium.）

3） **READ CAPACITY**：读取容量信息

4） **READ(10)**：回发在逻辑单元的数据，既回发 MBR(Main Boot Record)主引导扇区

5） **SENSE6**：目的在于获得设备内部很多潜在的信息，其中包括了是否设置了写保
（The MODE SENSE(6) command (see table 62) provides a means for a device server to report parameters to an application client. It is a complementary command to the MODE SELECT(6) command. Device servers that implement the MODE SENSE(6) command shall also implement the MODE SELECT(6) command.）

6） **WRITE(10)**：host 向 slave 发生数据并写在 u 盘存储器里面。

7） **TEST UNIT READY**：检查 U 盘准备好没有。

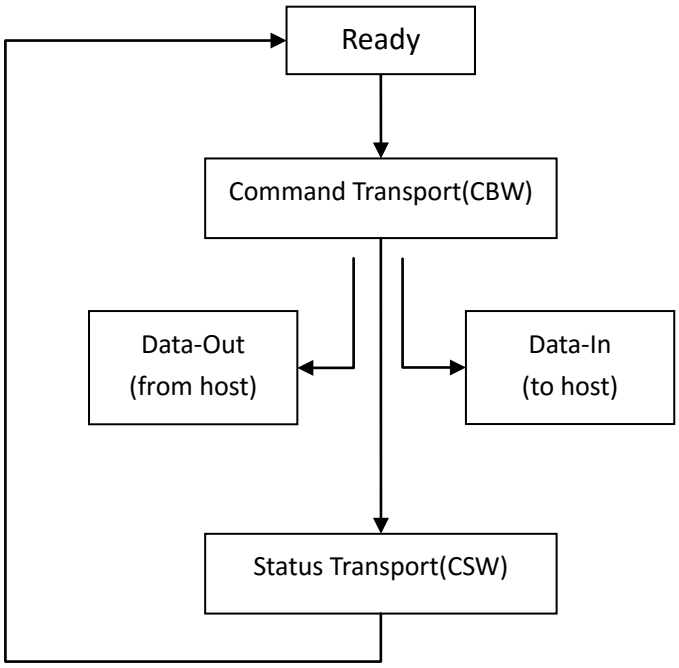
1.3、 Mass Storage 设备所使用的 SCSI 命令集(SCSI Commands)

指令代码		指令说明
0x00	Test Unit Ready	查询设备是否 ready
0x03	Request Sense	主机请求设备返回执行结果，及获取状态信息
0x04	Format Unit	格式化存储设备
0x12	Inquiry	获取设备信息
0x15	Mode Select(6)	允许 Host 对外部设备设置参数
0x1A	Mode Sense(6)	向 host 传输参数
0x1B	Start/Stop Uint	启动/停止存储单元电源（写保护）
0x1E	Prevent/Allow Medium Removal	禁止/允许存储介质移动
0x23	Read Format Capacity	查询当前容量及可用空间
0x25	Read Capacity(10)	读取设备容量
0x28	Read(10)	Host 从设备读取数据
0x2A	Write(10)	Host 写数据到存储设备
0x2F	Verify	在存储中验证数据
0x35	Sync Cache(10)	
0x55	Mode Select(10)	允许 Host 对外部设备设置参数
0x5A	Mode Sense(10)	向 host 传输参数
0x85	ATA Command Pass Through(16)	
0x91	Sync Cache(16)	
0x9E	Service Action In(16)	
0x9F	Service Action Out(16)	
0xA0	Report LUNs	索取设备的 LUN 数和 LUN 清单
0xA1	ATA Command Pass Through(12)	
0xA8	Read(12)	Host 从设备读取数据
0XA9	Service Action Out(12)	
0xAA	Write(12)	Host 写数据到存储设备
0xAB	Service Action In(12)	

1.4、U 盘的工程过程

设备插入到 USB 后，USB 即对设备进行搜索，并要求设备提供相应的描述符。在 USB Host 得到上述描述符后，即完成了设备的配置，识别出为 Bulk－Only 的 Mass Storage 设备，然后即进入 Bulk－Only 传输方式。在此方式下，USB 与设备间的所有数据均通过 Bulk－In 和 Bulk－Out 来进行传输，不再通过控制端点传输任何数据。

在这种传输方式下，有三种类型的数据在 USB 和设备之间传送，CBW、CSW 和普通数据。CBW（Command Block Wrapper，即命令块包）是从 USB Host 发送到设备的命令，命令格式遵从接口中的 bInterfaceSubClass 所指定的命令块，这里为 SCSI 传输命令集。USB 设备需要将 SCSI 命令从 CBW 中提取出来，执行相应的命令，完成以后，向 Host 发出反映当前命令执行状态的 CSW（Command Status Wrapper），Host 根据 CSW 来决定是否继续发送下一个 CBW 或是数据。Host 要求 USB 设备执行的命令可能为发送数据，则此时需要将特定数据传送出去，完毕后发出 CSW，以使 Host 进行下一步的操作。USB 设备所执行的操作可用下图描述：



1.5、CBW 命令块（Command Block Wrapper）

序号	域	字节	值	描述
0-3	dSignature	4	常量	常数 0x43425355，标识为 CBW 命令块
4-7	dCBWTag	4	数字	由主机发送的 CBW 标签。设备应该在相关的 CSW 的 dCSWTag 以相同的值应答主机。
8-11	dDataLength	4	数字	在本命令执行期间，主机期望通过 Bulk-In 或 Bulk-Out 端点传输的数据长度。如果为 0，则表示这之间没有数据传输。
12	bmFlags	1	数字	定义如下（Bit7 Direction（dDataLength 为 0 时，该值无意义）： Bit7=0：数据从主机到设备 Bit7=1：数据从设备到主机 Bit6：Obsolete 0 Bits 5..0：Reserved 0
13	bLUN	1	数字	表示正在发送命令字的设备的逻辑单元号（LUN）。对于支持多个 LUN 的设备，主机设置相对应的 LUN 值。否则，该值为 0。
14	bCBLength	1	数字	CB 的有效字节长度。有效值是在 1 到 16 之间。
15-30	CB[16]	16	数组	被设备解析执行的命令块。

1.6、CSW 状态块（Command Status Wrapper）

序号	域	字节	值	描述
0-3	dSignature	4	常量	常数 0x53425355，标识为 CSW 状态块
4-7	dCSWTag	4	数字	取相对应的 CBW 的 dCBWTag 值。
8-11	dDataResidue	4	数字	实际传输的数据个数和期望要传输的数据个数之差。
12	bStatus	1	数字	指示命令的执行状态。如果命令正确执行，bCSWStatus 返回 0，不正确返回 1，phase 错返回 2（当 HOST 收到此错误时需要对 Device 复位）

1.7、 USB 的描述符机制

1） bInterfaceClass= 0x08 //Mass Storage

2） bInterfaceSubClass= 0x06 //SCSI Transparent

3） bInterfaceProtocol= 0x50 //Bulk Only Transport

分别对应着 USB 的 Class,Subclass.protocol。

1.7.1、SCSI_Subclass 所支持的列表

Subclass	指令集	描述
0x00	SCSI command set not reported	De facto use
0x01	RBC	Allocated by USB-IF for RBC,RBC is defined of USB
0x02	MMC-5(ATAPI)	Allocated by USB-IF for MMC-5,MMC-5 is defined outside of USB
0x03	Obsolete	Was QIC-157
0x04	UFI	Specifies how to interface Floppy Disk Drives to USB
0x05	Obsolete	Was SFF-8070i
0x06	**SCSI transparent command set**	**Allocated by USB-IF for SCSI,SCSI standards are defined outside of USB**
0x07	LSD FS	LSDFS specified how host has to negotiate access before trying SCSI
0x08	IEEE 1667	Allocated by USB-IF for IEEE 1667,IEEE 1667 is defined outside of USB
0x09-0xFE	Reserved	Reserved
0xFF	Specific to device vendor	De facto use

1.7.1、SCSI_ protocol 所支持的列表

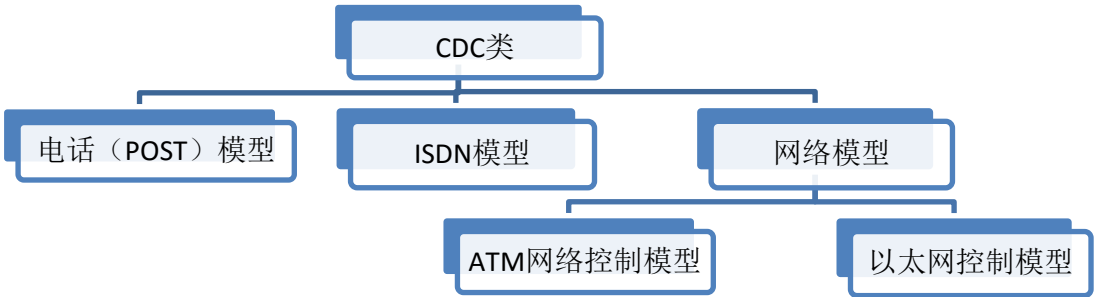
protocol	协议实现	描述
0x00	CBI(with command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt transport
0x01	CBI(with no command completion interrupt)	
0x02		
0x03-0x4F		
0x50	**BBB**	**UBS Mass Storage Class Bulk-Only(BBB) transport**
0x51-0x61		
0x62	SCSI transparent command set	
0x63-0xFE	LSD FS	
0xFF	IEEE 1667	

十一、 USB 人机接口（HID）

十二、 USB 虚拟串口（CDC）

USB 为了实现不同的应用， 将具有特定属性与服务的一类设备划分为一个 Class。如果提供相似格式的数据流或者相似的与主机交换方式，两个设备则被统一在一个 Class 中。 如 USB 标准就有 Audio Class、Communications Device Class、HIDClass、Video Class 等用于在 USB 接口上实现不同的设备接口。在 USB 标准协议中，有一类专用于通讯设备（主要包括电信通信设备和中速网络通信设备）的 CDC 协议，USB 的 CDC 类是 USB 通信设备类（Communication Device Class Specification ）的简称。可以通过 USB CDC 协议来将 USB 接口虚拟为其他通讯接口如串口，以太网接口，ISDN 接口等等。根据 CDC 协议所针对通信设备的不同，CDC 协议又被分成以下不同的模型： USB 传统电话业务（POTS ）模型， USB ISDN 模型和 USB 网络模型(如图所示)。本文就是通过 USB CDC 的网络模型来虚拟以太网接口。

USB CDC 通讯设备类结构:



CDC 协议由根据不同的功能可以分为三个部分：通讯设备类（Communication Devices Class） 、通讯接口类（Communication Interface Class） 、和数据接口类(Data Interface Class)。通讯设备类是设备层次的定义，通常用于标示一个通讯设备与该设备可以提供相应的接口。通讯接口类则定义了相应的通讯服务，包括如何对设备进行管理和控制，数据接口类则定义了如何传送数据。

在 USB CDC 协议中首先定义了以太网控制模型（ECM）用于配置与控制虚拟以太网接口，随后的补充协议中又定义了以太网仿真模型（EEM）用于封装与发送以太网包。

以太网控制模型

USB 以太网控制模型（ ECM）用于规范 USB 虚拟以太网接口配置与控制。 ECM 符合 CDC 协议，主要包括两个接口：数据接口与通讯接口。通讯接口类来用于配置与管理以太网的各种功能，主要包括 CDC 的枚举配置，
虚拟以太网接口的配置，报告虚拟以太网接口的状态。CDC 的枚举配置用于告知主机使用 CDC 来通讯，虚拟以太网接口的配置用于设置以太网的一些通用参数如组播、接收以太网包的过滤器、电源管理模式等。数据接口则用于在 USB 总线上交换 USB 数据包，这些 USB 数据包封装了完整的以太网包。需要注意的是 ECM 并没有明确规定如何对以太网包进行何种封装。