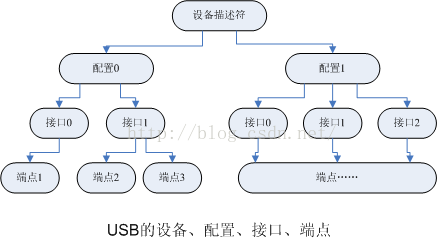
# USB协议

在终端用户看来，USB设备为主机提供了多种多样的附加功能，如文件传输，声音播放等，但对USB主机来说，它与所有USB设备的接口都是一致的。一个USB设备由3个功能模块组成：**USB总线接口**、**USB逻辑设备**和**功能单元**：

a -- 这里的USB总线接口指的是USB设备中的串行接口引擎（SIE）；

b -- USB逻辑设备被USB系统软件看作是**一个端点的集合**；

c -- **功能单元**被客户软件看作是一个**接口的集合**。SIE、端点和接口都是USB设备的组成单元；



设备通常有一个或多个配置；配置通常有一个或多个接口；

接口通常有一个或多个设置；接口有零或多个端点

## ****1、设备描述符****

设备代表一个USB设备，它由一个或多个配置组成。设备描述符用于说明设备的总体信息，并指明其所含的配置的个数。**一个USB设备只能有一个设备描述符。**

struct usb\_device\_descriptor

{

\_ \_u8 bLength; //描述符长度

\_ \_u8 bDescriptorType; //描述符类型编号

\_ \_le16 bcdUSB; //USB版本号

\_ \_u8 bDeviceClass; //USB分配的设备类code

\_ \_u8 bDeviceSubClass;// USB分配的子类code

\_ \_u8 bDeviceProtocol; //USB分配的协议code

\_ \_u8 bMaxPacketSize0; //endpoint0最大包大小

\_ \_le16 idVendor; //厂商编号

\_ \_le16 idProduct; //产品编号

\_ \_le16 bcdDevice; //设备出厂编号

\_ \_u8 iManufacturer; //描述厂商字符串的索引

\_ \_u8 iProduct; //描述产品字符串的索引

\_ \_u8 iSerialNumber; //描述设备序列号字符串的索引

\_ \_u8 bNumConfigurations; //可能的配置数量

} \_ \_attribute\_ \_ ((packed));

## ****2、配置描述符****

**一个USB设备可以包含一个或多个配置**，如USB设备的低功耗模式和高功耗模式可分别对应一个配置。在使用USB设备前，必须为其选择一个合适的配置。配置描述符用于说明USB设备中各个配置的特性，如配置所含接口的个数等。**USB设备的每一个配置都必须有一个配置描述符。**

struct usb\_config\_descriptor

{

\_ \_u8 bLength; //描述符长度

\_ \_u8 bDescriptorType; //描述符类型编号

\_ \_le16 wTotalLength; //配置所返回的所有数据的大小

\_ \_u8 bNumInterfaces; // 配置所支持的接口数

\_ \_u8 bConfigurationValue; //Set\_Configuration命令需要的参数值

\_ \_u8 iConfiguration; //描述该配置的字符串的索引值

\_ \_u8 bmAttributes; //供电模式的选择

\_ \_u8 bMaxPower; //设备从总线提取的最大电流

} \_ \_attribute\_ \_ ((packed));

## ****3、接口描述符****

**一个配置可以包含一个或多个接口**，例如对一个光驱来说，当用于文件传输时，使用其大容量存储接口；而当用于播放CD时，使用其音频接口。接口是端点的集合，可以包含一个或多个可替换设置，用户能够在USB处于配置状态时改变当前接口所含的个数和特性。接口描述符用于说明设备中各个接口的特性，如接口所属的设备类及其子类等。**USB设备的每个接口都必须有一个接口描述符**。

struct usb\_interface\_descriptor

{

\_ \_u8 bLength; //描述符长度

\_ \_u8 bDescriptorType; //描述符类型

\_ \_u8 bInterfaceNumber; // 接口的编号

\_ \_u8 bAlternateSetting; //备用的接口描述符编号

\_ \_u8 bNumEndpoints; //该接口使用的端点数，不包括端点0

\_ \_u8 bInterfaceClass; //接口类型

\_ \_u8 bInterfaceSubClass; //接口子类型

\_ \_u8 bInterfaceProtocol; //接口所遵循的协议

\_ \_u8 iInterface; //描述该接口的字符串索引值

} \_ \_attribute\_ \_ ((packed));

## ****4、端点描述符****

**端点是USB设备中的实际物理单元，USB数据传输就是在主机和USB设备各个端点之间进行的。**端点一般由USB接口芯片提供，例如Freescale公司的MC68HC908JB8和MC9S12UF32。USB设备中的每一个端点都有唯一的端点号，每个端点所支持的数据传输方向一般而言也是确定的：或是输入（IN），或是输出（OUT）。也有些芯片提供的端点的数据方向是可以配置的，例如MC68HC908JB8包含有两个用于数据收发的端点：端点1和端点2。其中端点1只能用于数据发送，即支持输入（IN）操作；端点2既能用于数据发送，也可用于数据接收，即支持输入（IN）和输出（OUT）操作。而MC9S12UF32具有6个端点。

 利用设备地址、端点号和传输方向就可以指定一个端点，并与它进行通信。端点的传输特性还决定了其与主机通信是所采用的传输类型，例如控制端点只能使用控制传输。根据端点的不同用途，可将端点分为两类：0号端点和非0号端点。

**0号端点比较特殊，它有数据输入IN和数据输出OUT两个物理单元，且只能支持控制传输。**所有的USB设备都必须含有一个0号端点，用作默认控制管道。USB系统软件就是使用该管道与USB逻辑设备进行配置通信的。0号端点在USB设备上电就存在并可以使用，而非0号端点必须要在配置以后才可以使用。

 根据具体应用的需要，USB设备还可以含有多个除0号端点以外的其他端点。对于低速设备，其附加的端点数最多为2个；对于全速/高速设备，**其附加的端点数最多为15个**。  
 struct usb\_endpoint\_descriptor

{

\_ \_u8 bLength; //描述符长度

\_ \_u8 bDescriptorType; //描述符类型

\_ \_u8 bEndpointAddress; //端点地址：0～3位是端点号，第7位是方向(0-OUT,1-IN)

\_ \_u8 bmAttributes; //端点属性：bit[0:1] 的值为00表示控制，为01表示同步，

//为02表示批量，为03表示中断

\_ \_le16 wMaxPacketSize; 本端点接收或发送的最大信息包的大小

\_ \_u8 bInterval;//轮询数据传送端点的时间间隔

//对于批量传送的端点以及控制传送的端点，此域忽略

//对于同步传送的端点，此域必须为1

\_ \_u8 bRefresh;

\_ \_u8 bSynchAddress;

} \_ \_attribute\_ \_ ((packed));

## ****5、字符串描述符****

**在USB设备中通常还含有字符串描述符，以说明一些专用信息**，如制造商的名称、设备的序列号等。它的内容以UNICODE的形式给出，且可以被客户软件所读取。对USB设备来说，**字符串描述符是可选的。**

struct usb\_string\_descriptor

{

\_ \_u8 bLength; //描述符长度

\_ \_u8 bDescriptorType; //描述符类型

\_ \_le16 wData[1];

} \_ \_attribute\_ \_ ((packed));

## ****6、管道****

在USB系统结构中，可以认为数据传输时在USB主机软件与USB设备的各个端点之间直接进行的，它们之间的连接称为管道。管道是在USB设备的配置过程中建立的。管道是对USB主机与USB设备间通信流的抽象，表示USB主机的数据缓冲区与USB设备的端点之间存在着逻辑数据传输，而实际的数据传输是由USB总线接口层来完成的。

管道与USB设备中的端点一一对应。一个USB设备含有多少个端点，其与USB主机进行通信时就可以使用多少条管道，且端点的类型决定了管道中数据的传输类型，例如中断端点对应中断管道，且该管道只能进行中断传输。不论存在着多少条管道，在各个管道中进行的数据传输都是相互独立的。

## ****7、USB端点分类****

USB 通讯的最基本形式是通过端点。一个USB端点只能向一个方向传输数据（从主机到设备(称为输出端点)或者从设备到主机(称为输入端点)）。端点可被看作一个单向的管道。

**USB 端点有 4 种不同类型, 分别具有不同的数据传送方式：**

### ****1) 控制CONTROL****

控制端点被用来控制对USB设备的不同部分访问. 通常用作配置设备、获取设备信息、发送命令到设备或获取设备状态报告。这些端点通常较小。每个 USB 设备都有一个控制端点称为"端点 0", 被 USB 核心用来在插入时配置设备。USB协议保证总有足够的带宽留给控制端点传送数据到设备.

### ****2) 中断INTERRUPT****

每当 USB 主机向设备请求数据时，中断端点以固定的速率传送小量的数据。此为USB 键盘和鼠标的主要的数据传送方法。它还用以传送数据到USB设备来控制设备。通常不用来传送大量数据。USB协议保证总有足够的带宽留给中断端点传送数据到设备.

### ****3) 批量BULK****

批量端点用以传送大量数据。这些端点通常比中断端点大得多. 它们普遍用于不能有任何数据丢失的情况。USB 协议不保证传输在特定时间范围内完成。如果总线上没有足够的空间来发送整个BULK包，它被分为多个包进行传输。这些端点普遍用于打印机、USB Mass Storage和USB网络设备上。

### ****4) 等时ISOCHRONOUS****

等时端点也批量传送大量数据, 但是这个数据不被保证能送达。这些端点用在可以处理数据丢失的设备中，并且更多依赖于保持持续的数据流。如音频和视频设备等等。

控制和批量端点用于异步数据传送，而中断和等时端点是周期性的。这意味着这些端点被设置来在固定的时间连续传送数据，USB 核心为它们保留了相应的带宽。

struct usb\_host\_endpoint{

struct usb\_endpoint\_descriptor desc;//端点描述符

struct list\_head urb\_list;//此端点的URB对列，由USB核心维护

void \*hcpriv;

struct ep\_device \*ep\_dev; /\* For sysfs info \*/

unsigned char\*extra;/\* Extra descriptors \*/

int extralen;

int enabled;

};

当调用USB设备驱动调用usb\_submit\_urb提交urb请求时，将调用int usb\_hcd\_link\_urb\_to\_ep(struct usb\_hcd \*hcd, struct urb \*urb)把此urb增加到urb\_list的尾巴上。(hcd: Host Controller Driver,对应数据结构struct usb\_hcd )

## ****8、常见的USB设备类别：****

1） Communications Device Class (CDC): （ACM类）在主机上产生一个虚拟COM端口

2）MSP430 USB开发人员包定义了四个子类型：

– Datapipe（类似于CDC的未格式化通用接口）  
- 鼠标  
–键盘  
–自定义

3）Mass Storage Class (MSC):当主机看到MSC接口时，它将开始从中安装存储卷。

## 9、USB设备参数查看方法

**1）调用libusb库中的函数进行查询，后续博客会提及**

**2）Linux下查看方法**

Linux系统下可查看USB设备信息的命令有：

**lsusb**

cat usb/devices

...

**3）Windows下查看方法**

Windows系统下可查看USB设备信息的方法有：

点击设备管理器，查看设备属性

**USB查看器usbtreeviewV3.1.0软件** 可查看USB设备详细信息

USB抓包工具--Bus Hound 可查看USB的通讯信息

# 一 libusb 介绍

libusb 设计了一系列的外部API 为应用程序所调用，通过这些API应用程序可以操作硬件，从libusb的源代码可以看出，这些API 调用了内核的底层接口，和kernel driver中所用到的函数所实现的功能差不多，只是libusb更加接近USB 规范。使得libusb的使用也比开发内核驱动相对容易的多。

# 二 libusb 的外部接口

## 2.1 核心初始化函数

### ****1) usb\_init****

函数定义： void usb\_init(void);

函数功能：libusb 初始化，这一步必须首先执行。

### ****2)usb\_find\_busses****

函数定义： int usb\_find\_busses(void);

寻找系统上的usb总线，任何usb设备都通过usb总线和计算机总线通信。此函数返回总线数，失败返回负值。

### 3) usb\_find\_devices

函数定义： int usb\_find\_devices(void);

寻找总线上的usb设备，这个函数必要在调用usb\_find\_busses()后使用。此函数成功返回设备数目，失败返回负值。

### ****4)usb\_get\_busses****

函数定义： struct usb\_bus \*usb\_get\_busses(void);

函数功能：返回全局变量 usb\_busses，用于遍历总线链表，查找总线 usb 设备。失败返回 NULL。

## ****2.2 设备操作函数****

### ****5）usb\_open****

函数定义： usb\_dev\_handle \*usb\_open(struct \*usb\_device dev);

函数功能：打开 usb 设备，引用的 usb\_dev\_handle 以及 usb\_device 结构体分别为usb设备操作句柄和usb设备描述符，具体定义可见头文件。函数返回 usb 设备的操作句柄，之后的对该 usb 设备的操作都通过该句柄进行。

### ****6) usb\_close****

函数定义： int usb\_close(usb\_dev\_handle \*dev);

函数功能：与usb\_open相对应，关闭设备，是必须调用的, 返回0成功，<0 失败。

### ****7) usb\_set\_configuration****

函数定义： int usb\_set\_configuration(usb\_dev\_handle \*dev, int configuration);

函数功能：设置当前设备使用的configuration(配置模式)，参数configuration 是你要使用的configurtation descriptoes中bConfigurationValue, 函数返回0成功，<0失败( 一个设备可能包含多个configuration,比如同时支持高速和低速的设备就有对应的两个configuration,详细可查看usb标准和具体的usb设备)

### ****8) usb\_claim\_interface****

函数定义： int usb\_claim\_interface(usb\_dev\_handle \*dev, int interface);

函数功能：注册与操作系统通信的接口，这个函数必须被调用，因为只有注册接口，才能做相应的操作（比如收发数据）。

Interface 指 bInterfaceNumber. (下面介绍的usb\_release\_interface 与之相对应，也是必须调用的函数)

### ****9) usb\_release\_interface****

函数定义： int usb\_release\_interface(usb\_dev\_handle \*dev, int interface);

函数功能：注销被usb\_claim\_interface函数调用后的接口，释放资源，和usb\_claim\_interface对应使用。程序结束时记得一定要注销接口，否则，下次注册接口时可能会报错。

### ****10) usb\_set\_altinterface****

函数定义： int usb\_set\_altinterface(usb\_dev\_handle \*dev, int alternate);

函数功能：设定当前接口中的活跃（起作用的）设置，即激活当前接口，alternate 为接口描述符中 bAlternateSetting 字段。成功返回0，失败返回负值。

### ****11)usb\_clear\_halt****

函数定义： int usb\_clear\_halt (usb\_dev\_handle \*dev, unsigned int ep);

函数功能：复位指定的endpoint，参数ep 是指bEndpointAddress。这个函数用来替代usb\_resetep函数。

注：libusb四种传输方式详细介绍可见http://blog.chinaunix.net/uid-25314474-id-3040231.html

## ****2.3 数据通讯****

### ****12)usb\_bulk\_write****

函数定义：int usb\_bulk\_write(usb\_dev\_handle \*dev, int ep, const char \*bytes, int size, int timeout);

函数功能：向usb设备批量写数据，\*dev为设备操作句柄，ep为传入指定的端点号，\*bytes为发送的数据地址，size为发送的数据长度，timeout 为超时时间（ms）。成功返回实际写入设备的字节数，失败返回负值。

### ****13) usb\_bulk\_read****

函数定义：int usb\_bulk\_read(usb\_dev\_handle \*dev, int ep, char \*bytes, int size, int timeout);

函数功能：批量读usb设备传过来的数据，\*dev为设备操作句柄，ep为传入指定的端点号，\*bytes为接收的数据地址，size为预计接收的数据长度，timeout 为超时时间（ms）。成功返回实际写入设备的字节数，失败返回负值。

### ****14) usb\_interrupt\_write****

函数定义：int usb\_interrupt\_write(usb\_dev\_handle \*dev, int ep, const char \*bytes, int size, int timeout);

函数功能：以中断的方式向usb设备写数据，\*dev为设备操作句柄，ep为传入指定的端点号，\*bytes为发送的数据地址，size为发送的数据长度，timeout 为超时时间（ms）。成功返回实际写入设备的字节数，失败返回负值。

### ****15) usb\_interrupt\_read****

函数定义：int usb\_interrupt\_write(usb\_dev\_handle \*dev, int ep, const char \*bytes, int size, int timeout);

函数功能：以中断的方式usb设备传过来的数据，\*dev为设备操作句柄，ep为传入指定的端点号，\*bytes为接收的数据地址，size为预计接收的数据长度，timeout 为超时时间（ms）。成功返回实际写入设备的字节数，失败返回负值。

### ****16) usb\_control\_msg****

函数定义：int usb\_control\_msg(usb\_dev\_handle \*dev, int requesttype, int request, int value, int index, char \*bytes, int size, int timeout);

函数功能：以控制请求的方式读写数据，从默认的管道发送和接受控制数据传入参数与 usb 协议保持一致。成功返回实际读写的字节数，失败返回负值。

### ****17) usb\_get\_string****

函数定义：int usb\_get\_string(usb\_dev\_handle \*dev, int index, int langid, char \*buf, size\_t buflen);

函数功能：获取设备的字符串描述，通过index和langid索引。返回Unicode字符串到buf中。返回实际写入buf的字节数，负数失败。

### ****18) usb\_get\_string\_simple****

函数定义：int usb\_get\_string\_simple(usb\_dev\_handle \*dev, int index, char \*buf, size\_t buflen);

函数功能：包装了 usb\_get\_string() 函数，返回指定index索引的字符串描述，并转换到C风格的ASCII。返回写入buf字节数，负数失败。

### ****19) usb\_get\_descriptor****

函数定义：int usb\_get\_descriptor(usb\_dev\_handle \*dev, unsigned char type, unsigned char index, void \*buf, int size);

函数功能：获取设备缺省控制管道的描述符，通过type和index索引。返回实际写入buf的字节数，负数失败。

### ****20) usb\_get\_descriptor\_by\_endpoint****

函数定义：int usb\_get\_descriptor(usb\_dev\_handle \*dev, unsigned char type, unsigned char index, void \*buf, int size);

函数功能：从设备获取描述符，以type和index索引，以ep标志的控制管道。返回读取字节数，负数失败。

## ****2.3 辅助函数****

### ****21) usb\_strerror****

函数定义：char \*usb\_strerror(void);

函数功能：根据错误返回值 errno，返回错误码码对应的字符串。

### ****22) usb\_set\_debug****

函数定义：void usb\_set\_debug(int level);

函数功能：设置 libusb 的调试级别，libusb 支持打印调试信息，level 值为指定的调试级别。

### ****23) usb\_device****

函数定义struct usb\_device \*usb\_device(usb\_dev\_handle \*dev);

函数功能：根据操作句柄返回对应的 usb\_device。

# 三 libusb 的接口函数相关参数设置

libusb 的接口函数相关参数是依据usb设备本身的硬件参数（如配置模式、接口地址等）所决定的，而查询这些参数的方式有很多，如调用libusb本身的查询函数，但我推荐一个更好的方法，用“USB查看器**usbtreeview**V3.1.0软件”可以清晰的获取usb本身的硬件设备。

本文第二部分所列举的函数只是libusb的一部分常用函数，想查看更多的函数，可上

http://libusb.sourceforge.net/api-1.0/api.html