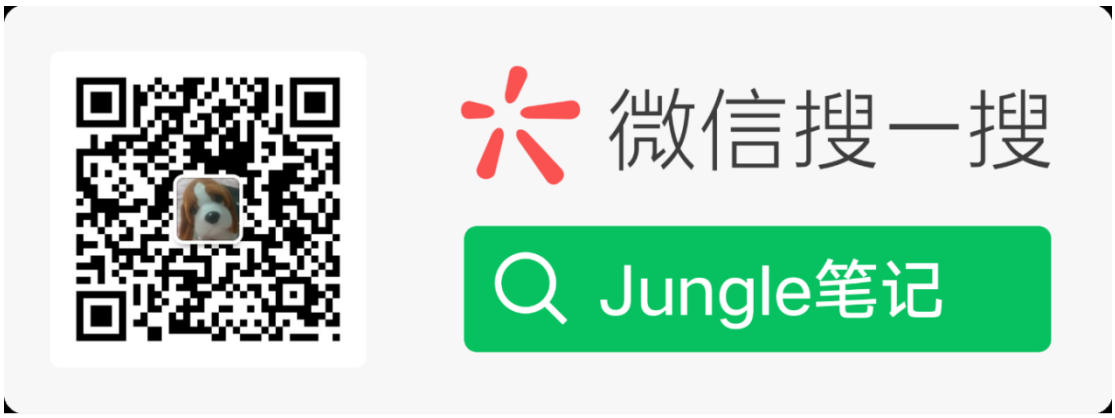
USB2.0 协议中文版整理

日期	版本	说明	修订人
2019/07/06	V1.0	初稿	FengJungle
2020/02/15	V1.1	1. 修改中文版章节结构 2. 增加 USB 协议实例 3. 完善电气规范部分	FengJungle



目录

1 简介.....5

1.1 Bus Topology.....5

1.1.1 USB host.....6

1.1.2 USB device.....6

1.2 USB 总线协议.....7

1.3 鲁棒性.....7

1.4 USB 设备的拔插.....7

2 USB 数据流模型.....8

2.1 Implementer Viewpoints.....8

2.2 USB 总线拓扑.....9

2.3 USB Communication Flow.....9

2.3.1 设备端点.....10

2.3.2 管道.....11

2.4 传输（Transfers）.....11

2.4.1 控制传输（Control Transfers）.....12

2.4.2 中断传输（Interrupt Transfers）.....15

2.4.3 批量传输（Bulk Transfers）.....16

2.4.4 同步传输（Isochronous Transfers）.....17

2.4.5 分离传输（Split Transaction）.....18

3 USB 物理规范和电气规范.....18

3.1 USB 线缆.....18

3.2 电气规范.....19

3.2.1 USB 的插入检测机制.....19

3.2.2 高速设备握手识别.....19

3.2.3 信号.....21

4 USB 通信协议.....26

4.1 字节顺序.....26

4.2 USB 数据格式.....27

4.2.1 域（Field）.....28
4.2.2 包（Packet）.....30
4.2.3 事务（Transaction）.....31
5 USB Device Framework.....34
5.1 USB 设备状态.....34
5.1.1 连接(Attached).....34
5.1.2 上电(Powered).....35
5.1.3 默认状态(Default).....35
5.1.4 地址(Address).....35
5.1.5 配置状态(Configured).....36
5.1.6 挂起状态(Suspended).....36
5.2 总线枚举(Bus Enumeration).....36
5.3 USB 请求处理的限制.....38
5.4 设备请求.....38
5.5 标准设备请求.....40
5.5.1 Clear Feature.....42
5.5.2 Get Configuration.....42
5.5.3 Get Descriptor.....42
5.5.4 Get Interface.....44
5.5.5 Get Status.....44
5.5.6 Set Address.....45
5.5.7 Set Configuration.....46
5.5.8 Set Descriptor.....47
5.5.9 Set Feature.....47
5.5.10 Set Interface.....48
5.6 描述符(Descriptor).....48
5.6.1 设备（Device）.....48
5.6.2 设备限定（Device_Qualifier）.....50
5.6.3 配置（Configuration）.....51
5.6.4 其他速度配置（Other_Speed_Configuration）.....52

5.6.5 接口（Interface）.....52
5.6.6 端点（Endpoint）.....53
5.6.7 字符串（String）.....54
6 USB Hub.....55
7 参考资料.....55

1 简介

本章对应于 USB2.0 协议第 4 章。

USB（Universal Serial Bus）是一种支持热插拔的高速串行传输总线，它使用差分信号来传输数据。在 USB1.0 和 USB1.1 版本中，只支持 1.5Mb/s 的低速（low-speed）模式和 12Mb/s 的全速（full-speed）模式，在 USB 2.0 中，又加入了 480Mb/s 的高速模式，USB3.0(super speed)，传输速率最大 5Gbps。USB2.0 被设计成为向下兼容的模式，当有全速（USB 1.1）或者低速（USB 1.0）设备连接到高速（USB 2.0）主机时，主机可以通过分离传输来支持它们。一条 USB 总线上，可达到的最高传输速度等级由该总线上最慢的“设备”决定。

USB 体系包括 USB host（主机）、USB device(设备)以及物理连接(USB interconnect) 三个部分。其中，设备(USB device) 又分为 USB function 和 USB Hub。USB interconnect 是 USB 设备连接到主机并与之通信的方式。

1.1 Bus Topology

USB 上的设备通过分层的星形拓扑物理连接到主机，如下图所示。USB 连接点由称为集线器的特殊类别的 USB 设备提供。集线器提供的附加连接点称为端口。主机包括称为根集线器的嵌入式集线器。主机通过根集线器提供一个或多个连接点。为主机提供附加功能的 USB 设备称为功能。为了防止循环附件，USB 层的星形拓扑结构上采用了分层排序。

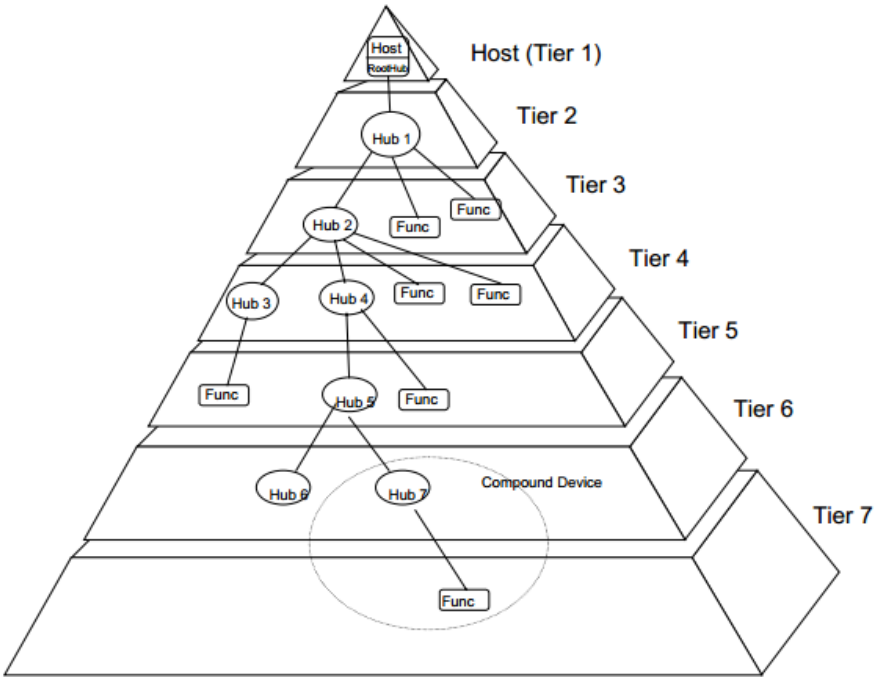


图 1 Bus Topology（USB2.0 Fig4-1）

以 HOST-ROOT HUB 为起点，最多支持 7 层（Tier），也就是说任何一个 USB 系统中最多可以允许 5 个 USB HUB 级联。一个复合设备（Compound Device）将同时占据两层或更多的层。

1.1.1 USB host

任何 USB 系统中只有一个主机。主机系统的 USB 接口被称为主机控制器。主机控制器可以以硬件，固件或软件的组合来实现。根集线器集成在主机系统内以提供一个或多个连接点。USB Host 通过 Host Controller 与 USB device 交互。Host 主要负责：

- 检测 USB 设备拔插
- 管理 Host 和 Device 之间的控制流、数据流
- 收集 USB 总线状态和活动数据信息
- 为连入 USB 总线的设备供电

1.1.2 USB device

USB device 可以分为 USB hub 和 USB function。

1.1.2.1 USB Hub

USB HUB 提供了一种低成本、低复杂度的 USB 接口扩展方法。HUB 的上行 PORT 面向 HOST，下行 PORT 面向设备(HUB 或功能设备)。在下行 PORT 上，HUB 提供了设备连接检测和设备移除检测的能力，并给各下行 PORT 供电。HUB 可以单独使能各下行 PORT。不同 PORT 可以工作在不同的速度等级(高速/全速/低速)。

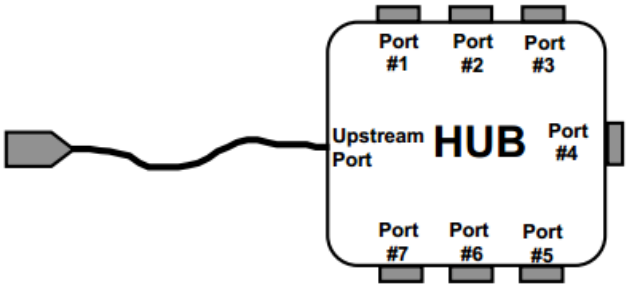


Figure 4-3. A Typical Hub

一个 USB HOST 最多可以同时支持 128 个地址，地址 0 作为默认地址，只在设备枚举期间临时使用，而不能被分配给任何一个设备。因此一个 USB HOST 最多可以同时支持 127 个地址，如果一个设备只占用一个地址，那么可最多支持 127 个 USB 设备。在实际的 USB 体系中，如果要连接 127 个 USB 设备，必须要使用 USB HUB，而 USB HUB 也是需要占用地址的，所以实际可支持的 USB 功能设备的数量将小于 127。

ROOT HUB 是一个特殊的 USB HUB，它集成在主机控制器里，不占用地址。ROOT HUB

不但实现了普通 USB HUB 的功能，还包括其他一些功能。

“复合设备（Compound Device）”可以占用多个地址。所谓复合设备其实就是把多个功能设备通过内置的 USB HUB 组合而成的设备，比如带录音话筒的 USB 摄像头等。

1.1.2.2 USB function

能够通过总线传输或接收数据或控制信息的设备，在 USB2.0 标准中，规范中有详细的章节进行定义。主要有以下三类：

- A human interface device such as a mouse, keyboard, tablet, or game controller
- An imaging device such as a scanner, printer, or camera
- A mass storage device such as a CD-ROM drive, floppy drive, or DVD drive

1.2 USB 总线协议

USB 是一种轮询总线，由 Host 发起所有的数据传输。

大部分总线传输包含 3 个包（packet）。每个传输都由 Host 先发出令牌包（Token Packet），明确传输类型、传输方向、USB 设备地址和端点号。对应地址的 USB 设备接收并解析包。一次传输可以由 Host 发向设备，也可以由设备发送至 Host，方向由令牌包说明。传输中的数据过程是可选的，即有的传输没有数据过程。在低速/全速设备中，一次传输由 4 个包组成。

1.3 鲁棒性

USB 的鲁棒性通过下述几点来保证：

- 使用差分驱动、接收器保证信号完整性
- 控制和数据传输过程中 CRC 校验
- 设备插拔检测
- 超时机制，检测丢失或损坏的包
- 流式数据的流量控制：保证同步和硬件缓冲管理

软件和硬件层面都可以进行错误处理。硬件上的故障处理机制包括，对一次失败的传输，USB Host controller 可以 retry 最多 3 次。

1.4 USB 设备的拔插

USB 设备支持热插拔。USB 设备通过 HUB 连接到 USB 总线。Hub 有一个状态位，标记 USB 设备是插入还是拔出（Attachment or removal）。Host 会查询 Hub 端口的状态位，当

检测到设备插入时，Host 将使能该 Hub 对应的 port，并通过默认地址（0）从设备控制管道为设备分配地址。当一个设备从 Hub 的某个 port 拔出时，Hub 将禁能（Disable）该 port 并报告给 Host。

2 USB 数据流模型

本章对应 USB2.0 协议第 5 章。

2.1 Implementer Viewpoints

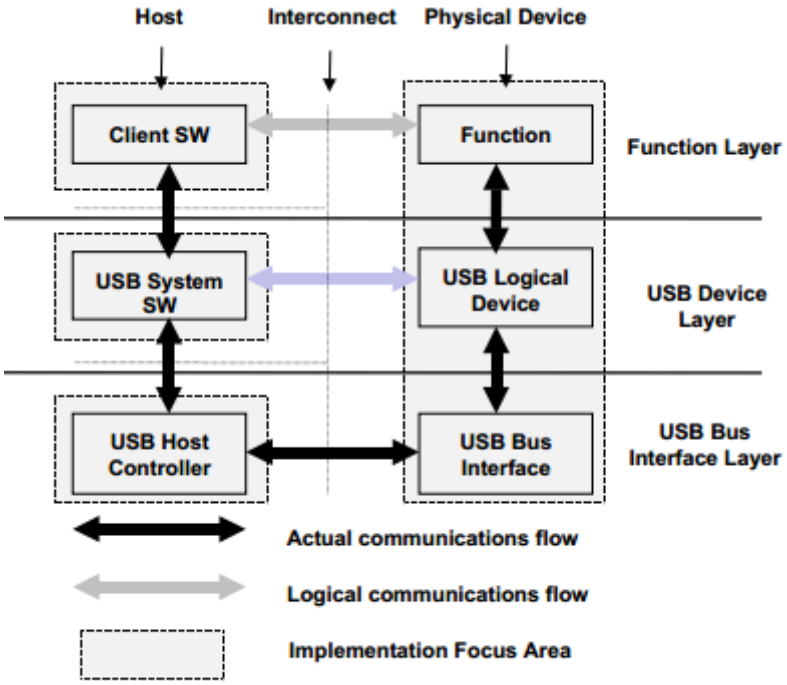


Figure 5-2. USB Implementation Areas

- USB Physical Device：具有一定功能的 USB 设备硬件
- Client Software：在 Host 上运行的与 USB Device 通讯的软件，一般与具体的 USB 设备配套
- USB System Software：USB 系统软件，一般与操作系统配套提供，与具体的 USB 设备和 Client Software 无关；
- USB Host Controller：包括软硬件，允许 USB 设备连接到 Host。

从 Figure5-2 可看出，在 Host 和 USB Device 之间建立的连接需要通过不同的层（Layer）和经过不同的实体（Entities）。USB Bus Interface 提供了物理、信号和包的连接；USB Device 层负责 USB 系统软件与设备交互。功能层为 USB 设备与 Host 交互匹配合适的 Client Software。

2.2 USB 总线拓扑

物理上，USB 设备通过 HUB 连接到 USB 总线上，此时 HUB 上的接入点称为端口（Port）。Host 有一个内嵌的 Hub，称为 Root Hub。USB 设备和 Host 物理上的连接形成了一个星形拓扑，如 Figure5-5。连接到 Hub 上的设备可能是单个设备，也可以是复合设备（Compound Device）。从 Host 的角度来看，复合设备和单个设备是一样的。每一个 USB 设备提供不同的功能，又被称作 Function。

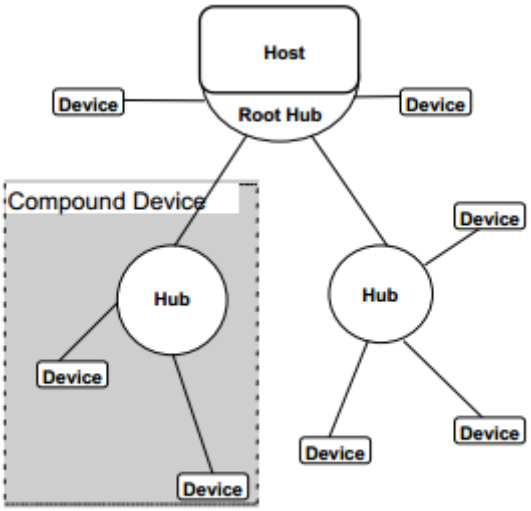


Figure 5-5. USB Physical Bus Topology

尽管物理上整个 USB 系统的分布呈星形拓扑，但逻辑上，每个 USB Device 是直接连接到 Host 端，如 Figure5-7。Hub 也是 Logical Device，不过没有在 Figure5-7 中表示出来。当一个 Hub 被移除了，连接在这个 Hub 上的所有 Device 都被移除。

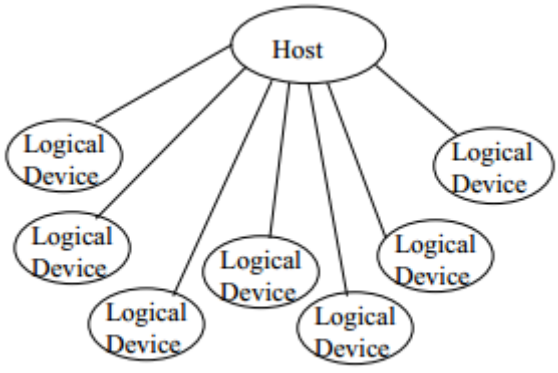


Figure 5-7. USB Logical Bus Topology

2.3 USB Communication Flow

从 USB 系统角度而言，一个逻辑上的 USB 设备是一个端点（Endpoint）的集合。分组

的端点构成一个接口（Interface）。USB 系统通过默认控制管道来管理设备。Client Software 使用管道（Pipe）来管理接口，通过 Host 上的 Buffer 和 USB 设备上的端点来请求数据。Host Controller 打包数据并将数据包发送出去，如 Figure5-10。

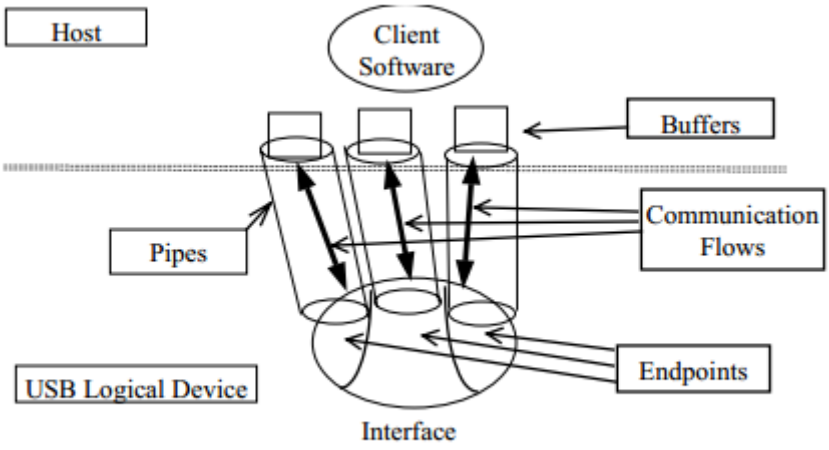


Figure 5-10. USB Communication Flow

2.3.1 设备端点

端点（Endpoint）是 USB 设备上可被独立识别的端口，是 Host 和 Device 通讯流的逻辑终点，是 USB 设备中可以进行数据收发的最小单元。一系列相互独立的端点在一起构成了 USB 逻辑设备。当设备连入 USB 总线时会被分配一个唯一的地址，设备上每一个端点有唯一的端点号。设备可决定每个端点的数据传输方向（输入或输出）。因此，**<设备地址，端点号，数据传输方向> 使得每个端点可被唯一寻址**。每个端点的属性包括总线访问频率、带宽、端点号、最大包长度、传输类型和数据传输方向（见端点描述符 5.6.6）。

2.3.1.1 0 端点

每个 USB 设备必须要有一个端点 0，其作用为对设备枚举和对设备进行一些基本的控制功能，端点 0 也被称为**控制端点**。并且它与其他的端点还有一个不同之处在于端点 0 的数据传输方向是双向的，即端点 0 既可以给主机发送数据，也可以接收主机发送过来的数据，而其它端点均为单向。

2.3.1.2 非 0 端点

除了控制端点以外，每个 USB 设备允许有一个或多个非 0 端点。低速设备最多只有两个非 0 端点。高速和全速设备最多支持 15 组端点。除了端点 0，其余的端点在设备配置之前不能与主机通信，只有向主机报告这些端点的特性并被确认后才能被激活。

按照 USB 协议栈的层次划分：

- 一个 Host 可能有一个或者多个 Device
- 一个 Device 可能有一个或者多个 Interface
- 一个 Interface 可能有一个或者多个 Endpoint

2.3.2 管道

管道（Pipe）是主机和设备端点之间数据传输的模型，共有两种类型的管道：无格式的流管道（Stream Pipe）和有格式的信息管道（Message Pipe）。

- **流管道**：数据从流通道一端流进的顺序与它们从流通道另一端流出时的顺序是一样的（先进先出），并且流通道中的通信流总是单向的。
- **信息管道**：信息管道与端点的关系同流通道与端点的关系是不同的。首先，主机向 USB 设备发出一个请求;接着，就是数据的传送;最后，是一个状态阶段（这部分即一次命令请求的过程）。为了能够容纳请求/数据/状态的变化，信息管道要求数据有一个格式，此格式保证了命令能够被可靠地传送和确认。信息管道允许双方向的信息流。

任何 USB 设备一旦上电就存在一个信息管道，即默认的控制管道，USB 主机通过该管道来获取设备的描述、配置、状态，并对设备进行配置。

2.4 传输（Transfers）

USB 系统中的数据传输，宏观看是在 HOST 和 USB 功能设备之间进行。微观看是在应用软件的 Buffer 和 USB 功能设备的端点之间进行。**一般来说端点都有 Buffer，可以认为 USB 通讯就是应用软件 Buffer 和设备端点 Buffer 之间的数据交换，交换的通道称为管道。**通常需要多个管道来完成数据交换，因为同一管道只支持一种类型的数据传输。用在一起来对设备进行控制的若干管道称为设备的接口，这就是端点、管道和接口的关系。

USB 控制传输分为以下四种：

批量传输：批量传输一般用于批量的和非实时的数据传输，通俗的来说就是用于数据量大但对时间要求又不高的场合的一种传输方式，类似用于 USB 打印机和 USB 扫描仪等。

中断传输：中断传输一般用于小批量的和非连续的数据传输，通俗的来说就是用于数据量小、数据不连续的但实时性高的场合的一种传输方式，类似用于 USB 鼠标和 USB 键盘等。

等时传输：等时传输也有“同步传输”的叫法，一般用于要求数据连续、实时且数据量大的场合，其对传输延时十分敏感，类似用于 USB 摄像设备，USB 语音设备等。

控制传输：控制传输是一种特殊的传输方式，且传输过程相对以上三种而言更复杂一些，

但也十分重要。当 USB 设备初次连接主机时，用控制传输传送控制命令等对设备进行配置。同时设备接入主机时，需要通过控制传输去获取 USB 设备的描述符以及对设备进行识别，在设备的枚举过程中都是使用控制传输进行数据交换。

USB 采用“令牌包”-“数据包”-“握手包”的传输机制，在令牌包中指定数据包去向或者来源的设备地址和端点(Endpoint)，从而保证了只有一个设备对被广播的数据包/令牌包作出响应。握手包表示了传输的成功与否。

USB 采用**轮询的广播机制**传输数据，所有的传输都由主机发起，任何时刻整个 USB 体系内仅允许一个数据包的传输，即不同物理传输线上看到的数据包都是同一被广播的数据包。

各种传输类型下，对于带宽、包长、速率限制、传输错误管理等如下：

	Control			Bulk			Interrupt			isoch		
	HS	FS	LS	HS	FS	LS	HS	FS	LS	HS	FS	LS
带宽	保证			没有保证			有限的保留带宽			保证、传输率固定		
	20%	10%	10%				23.4M B/s	62.5 K B/s	800 B/s		999 K B/s	2.9M B/s
最大数据包长度	64	64	8	512	64	N.A	1024	64	8	1024	1023	N.A
传输错误管理	握手包、PID翻转			握手包、PID翻转			握手包、PID翻转			无错误纠察		
组成	Setup stage + Optional Data stage + Status stage			一个或多个Data transaction (IN or OUT)			一个Data transaction (IN)			一个或多个Data transaction (IN or OUT)		

2.4.1 控制传输（Control Transfers）

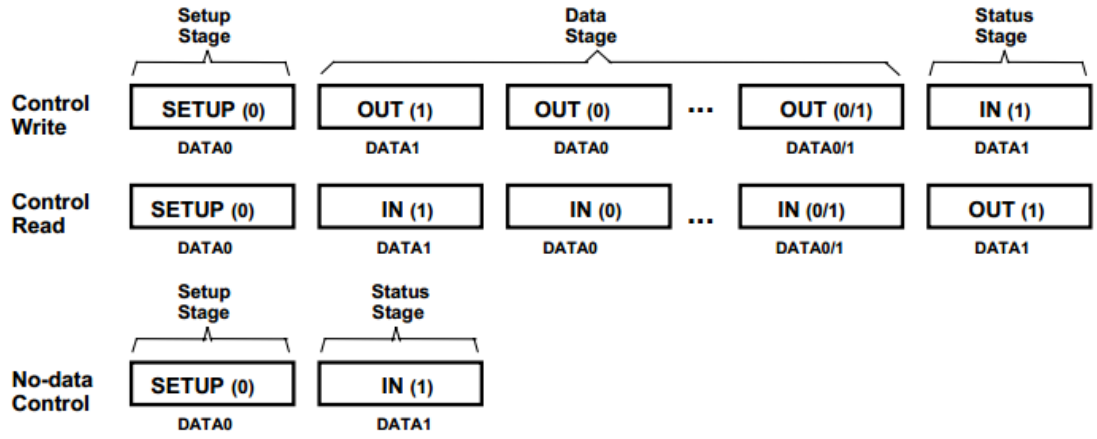
控制传输是一种可靠的双向传输，是最重要也是最复杂的。一次控制传输分为三个(或两个)阶段：建立(Setup)、数据(DATA)(可能没有)以及状态(Status)。每个阶段都由一次或多次(数据阶段)事务(Transaction)传输组成。在 USB 设备初次接到主机后，主机通过控制传输来交换信息、设备地址和读取设备的描述符，使得主机识别设备，并安装相应的驱动程序。控制传输是双向的传输，必须有 IN 和 OUT 两个方向上的特定端点号的控制端点来完成两个方向上的控制传输。

2.4.1.1 建立阶段

主机从 USB 设备获取配置信息，并设置设备的配置值。建立阶段的数据交换包含了 SETUP 令牌包、紧随其后的 DATA0 数据包以及 ACK 握手包(**SETUP 只能使用 DATA0 包，8 字节**)。它的作用是执行一个设置（概念含糊）的数据交换，并定义此控制传输的内容(即：在 Data Stage 中 IN 或 OUT 的 data 包个数，及发送方向，在 Setup Stage 已经被设定)。建立阶段，Device 只能返回 ACK 包，或者不返回任何包。

2.4.1.2 数据阶段

数据过程是可选的。一个数据过程包含一笔或者多笔数据事务。数据过程的第一个数据包必须是 DATA1 包，然后每次正确传输一个数据包就在 DATA0 和 DATA1 之间交替。



控制读/控制写包序列（Fig8-37）

根据数据阶段的数据传输的方向，控制传输又可分为 3 种类型：

- 控制读取（读取 USB 描述符）： 是将数据从设备读到主机上，读取的数据是 USB 设备描述符。该过程如上图的【Control Read】所示。对每一个数据信息包，首先，主机会发送一个 IN 令牌信息包，表示要读数据进来。然后，设备将数据通过 DATA1/DATA0 数据信息包回传给主机。最后，主机将以下列的方式加以响应：当数据已经正确接收时，主机送出 ACK 令牌包；当主机正在忙碌时，发出 NAK 握手包；当发生了错误时，主机发出 STALL 握手包。
- 控制写入（配置 USB 设备）： 是将数据从主机传到设备上，所传的数据即为对 USB 设备的配置信息，如上图的【Control Wirte】所示。对每一个数据信息包，主机将会送出一个 OUT 令牌信息包，表示数据要送出去。紧接着，主机将数据通过 DATA1/DATA0 数据信息包传递至设备。最后，设备将以下列方式加以响应：当数据已经正确接收时，设备送出 ACK 令牌信息包；当设备正在忙碌时，设备发出 NAK 握手信息包；当发生了错误时，设备发出 STALL 握手信息包。
- 无数据控制： 如上图

2.4.1.3 状态阶段

用来表示整个传输的过程已完全结束。通过一次 IN/OUT 传输表明请求是否成功完成。状态阶段传输的方向必须与数据阶段的方向相反，即原来是 IN 令牌包，这个阶段应为 OUT 令牌包；反之，原来是 OUT 令牌包，这个阶段应为 IN 令牌包。

对于【控制读取】而言，主机会送出 OUT 令牌包，其后再跟着 0 长度的 DATA1 包。

而此时，设备也会做出相对应的动作，送 ACK 握手包、NAK 握手包或 STALL 握手包。

相对地，对于【控制写入】传输，主机会送出 IN 令牌包，然后设备送出表示完成状态阶段的 0 长度的 DATA1 包，主机再做出相对应的动作：送 ACK 握手包、NAK 握手包或 STALL 握手包。

2.4.1.4 最大包长度限制

控制传输通过控制管道在应用软件和 Device 的控制端点之间进行，控制传输过程中传输的数据是有格式定义的，USB 设备或主机可根据格式定义解析获得的数据含义。其他三种传输类型都没有格式定义。控制传输对于最大包长度有固定的要求。对于高速设备该值为 64Byte，对于低速设备该值为 8，全速设备可以是 8 或 16 或 32 或 64。

最大包长度表征了一个端点单次接收/发送数据的能力，实际上反应的是该端点对应 Buffer 的大小。Buffer 越大，单次可接收/发送的数据包越大，反之亦反。当通过一个端点进行数据传输时，若数据的大小超过该端点的最大包长度时，需要将数据分成若干个数据包传输。并保证除最后一个包外，所有的包长度均等于该最大包长度。这也就是说如果一个端点收到/发送了一个长度小于最大包长度的包，即意味着数据传输结束。

2.4.1.5 访问总线限制

控制传输在访问总线时也受到一些限制,如高速端点的控制传输不能占用超过 20%的微帧，全速和低速的则不能超过 10%。在一帧内如果有多余的未用时间，并且没有同步和中断传输，可以用来进行控制传输。

与批量传输相比，在流程上并没有多大区别，区别只在于该事务传输发生的端点不一样、支持的最大包长度不一样、优先级不一样等这样一些对用户来说透明的东西。

2.4.1.6 控制传输实例

下图是一个控制读取的控制传输过程实例，每一个过程数据的含义可参看 5.5 和 5.6.

建立阶段		SYNC	SETUP	ADDR	ENDP	CRC5			
	令牌包	00000001	0xB4	0x00	0x00	0x08			
		SYNC	DATA0	DATA			CRC16		
	数据包	00000001	0xC3	80 09 00 01 00 00 40 00			0xBB29		
		SYNC	ACK						
	握手包	00000001	0x4B						
数据阶段		SYNC	IN	ADDR	ENDP	CRC5			
	令牌包	00000001	0x96	0x00	0x00	0x08			
		SYNC	DATA1	DATA					CRC16
	数据包	00000001	0xD2	12 01 00 01 DC 00 00 10 71 04 66 06 00 01 00 00					0x42C6
		SYNC	ACK						
	握手包	00000001	0x4B						
状态阶段		SYNC	OUT	ADDR	ENDP	CRC5			
	令牌包	00000001	0x87	0x00	0x00	0x08			
		SYNC	DATA1	DATA	CRC16				
	数据包	00000001	0xD2		0x0000				
		SYNC	ACK						
	握手包	00000001	0x4B						

2.4.2 中断传输（Interrupt Transfers）

中断传输是一种**轮询**的传输方式，是一种**单向**的传输。HOST 通过固定的间隔对中断端点进行查询，若有数据传输或可以接收数据则返回数据或发送数据。否则返回 NAK，表示尚未准备好。中断传输的延迟有保证，但并非实时传输，它是一种延迟有限的可靠传输，支持错误重传。对于高速/全速/低速端点，最大包长度分别可以达到 1024/64/8 Bytes。高速中断传输不得占用超过 80%的微帧时间，全速和低速不得超过 90%。 中断端点的轮询间隔由在**端点描述符**中定义，全速端点的轮询间隔可以是 1 ~ 255ms；低速端点为 10 ~ 255ms；高速端点为(2^interval-1)*125us，其中 interval 取 1 到 16 之间的值。

主机在排定中断传输任务时，会根据对应中断端点描述符中指定的查询间隔发起中断传输。中断传输有较高的优先级，仅次于同步传输。同样中断传输也采用 PID 翻转的机制来保证收发端数据同步。

中断传输方式总是用于对设备的查询，以确定是否有数据需要传输。因此中断传输的方向总是从 USB 设备到主机。

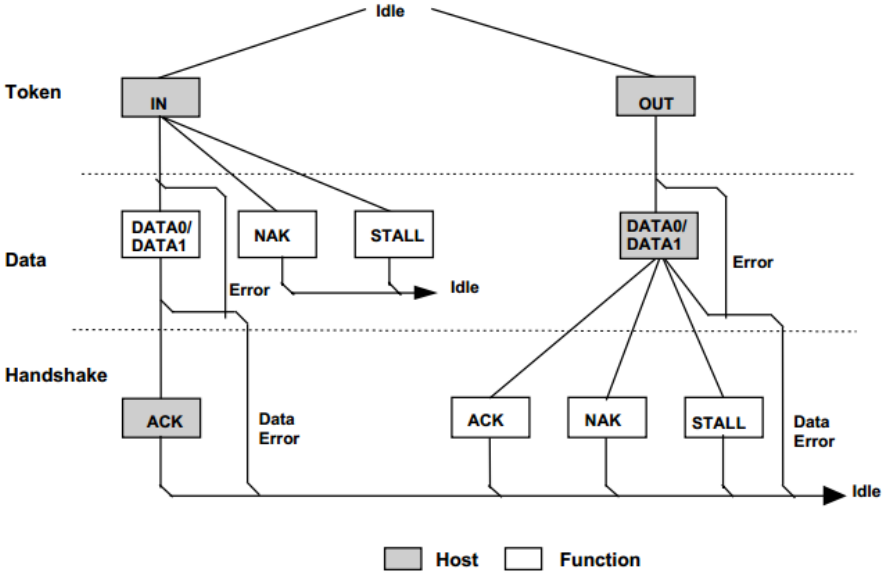
除高速高带宽中断端点外，一个微帧内仅允许一次中断事务传输。高速高带宽端点最多可以在一个微帧内进行三次中断事务传输，传输高达 3072 字节的数据。

所谓单向传输，并不是说该传输只支持一个方向的传输。而是指在某个端点上该传输仅支持一个方向，或输出、或输入。如果需要在两个方向上进行某种单向传输，需要占用两个端点，分别配置成不同的方向，可以拥有相同的端点编号。

中断传输由 OUT 事务和 IN 事务构成，用于键盘、鼠标等 HID 设备的数据传输。

中断传输在流程上除不支持 PING 之外，其他的跟批量传输是一样的。他们之间的区别也仅在于事务传输发生的端点不一样、支持的最大包长度不一样、优先级不一样等这样一些对用户来说透明的东西。

下图为中断传输的流程图。



2.4.3 批量传输（Bulk Transfers）

批量传输由 OUT 事务和 IN 事务构成，是一种可靠的单向传输，但延迟没有保证，它尽量利用可以利用的带宽来完成传输，适合数据量比较大的传输。低速 USB 设备不支持批量传输，高速批量端点的最大包长度为 512，全速批量端点的最大包长度可以为 8、16、32、64。用于传输大量数据，要求传输不能出错，但对时间没有要求，适用于打印机、存储设备等

批量传输在访问 USB 总线时，相对其他传输类型具有最低的优先级，USB HOST 总是优先安排其他类型的传输，当总线带宽有富余时才安排批量传输。高速的批量端点必须支持 PING 操作，向主机报告端点的状态。NYET 表示否定应答，没有准备好接收下一个数据包，ACK 表示肯定应答，已经准备好接收下一个数据包。

它通过在硬件级执行“错误检测”和“重传”来确保 host 与 device 之间“准确无误”地传输数据，即可靠传输。它由三种包组成(即 IN 事务或 OUT 事务)：Token、Data 和 Handshake。

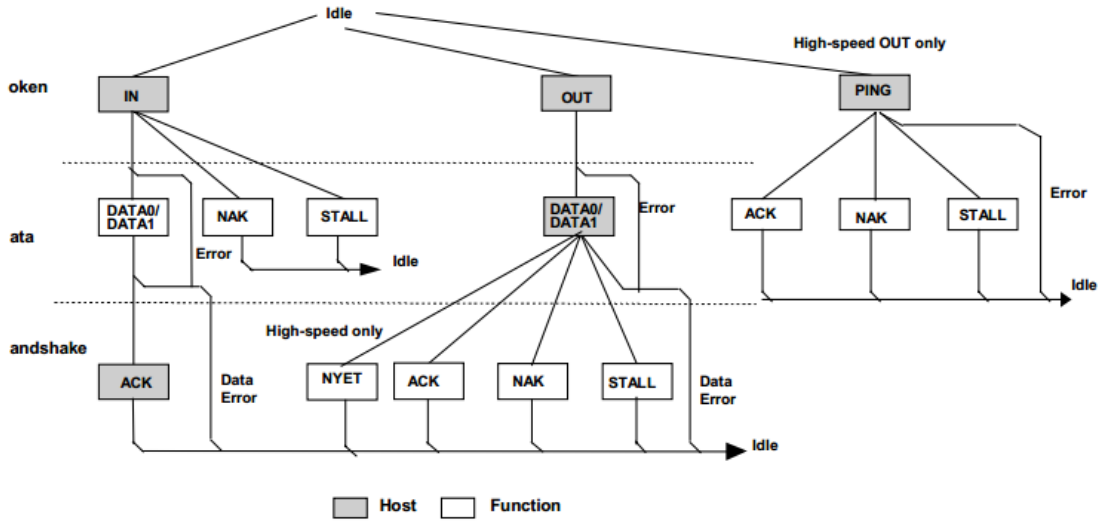
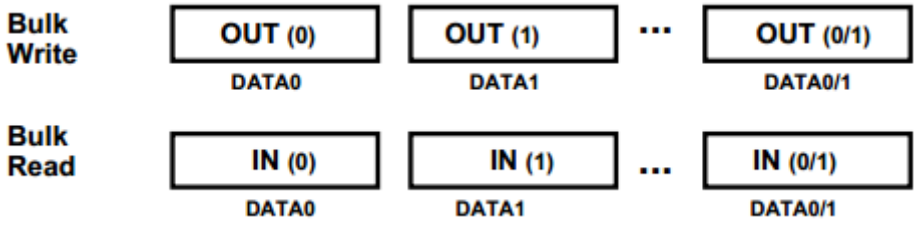


图 2　批量传输格式（Fig8-30）

若数据量比较大，将采用多次批量事务传输来完成全部数据的传输，传输过程中数据包的 PID 按照 DATA0-DATA1-DATA0-…的方式翻转，以保证发送端和接收端的同步。若成功则将错误次数计数器清 0，否则累加该计数器。

USB 允许连续 3 次以下的传输错误，错误时会重试该传输，若成功则将错误次数计数器清零，否则累加该计数器。超过三次后，HOST 认为该端点功能错误（STALL），放弃该端点的传输任务。

一次批量传输(Transfer)由 1 次到多次批量事务传输(Transaction)组成。



上图（USB2.0 规范的 8.5.2 章节）为批量读写时数据位和 PID 的变化情况。主机总是使用配置事件将总线传输的第一个事务初始化为 DATA0 PID。 第二个事务使用 DATA1 PID，并且在批量传输的其余部分中交替使用 DATA0 和 DATA1 传输数据。翻转同步：发送端按照 DATA0-DATA1-DATA0-…的顺序发送数据包，只有成功的事务传输才会导致 PID 翻转，也就是说发送段只有在接收到 ACK 后才会翻转 PID，发送下一个数据包，否则会重试本次事务传输。同样，若在接收端发现接收到的数据包不是按照此顺序翻转的，比如连续收到两个 DATA0，那么接收端认为第二个 DATA0 是前一个 DATA0 的重传。

2.4.4　同步传输（Isochronous Transfers）

同步传输是一种实时的、不可靠的传输，不支持错误重发机制。只有高速和全速端点支

持同步传输，高速同步端点的最大包长度为 1024，低速的为 1023。由 OUT 事务和 IN 事务构成。有两个特殊地方：

第一，在同步传输的 IN 和 OUT 事务中是没有返回包阶段的；

第二，在数据包阶段所有的数据包都为 DATA0。同步传输由 Token 和 Data 两种包组成。

同步传输不支持“handshake”和“重传能力”，所以它是不可靠传输。

同步传输适用于必须以固定速率抵达或在指定时刻抵达，可以容忍偶尔错误的数据上。实时传输一般用于麦克风、喇叭、UVC Camera 等设备。同步传输有最高的优先级

除高速高带宽同步端点外，一个微帧内仅允许一次同步事务传输，高速高带宽端点最多可以在一个微帧内进行三次同步事务传输，传输高达 3072 字节的数据。全速同步传输不得占用超过 80%的帧时间，高速同步传输不得占用超过 90%的微帧时间。同步端点的访问也和中断端点一样,有固定的时间间隔限制。

2.4.5 分离传输（Split Transaction）

分离传输是在主机控制器和 USB HUB 之间的传输，它仅在主机控制器和 HUB 之间执行，通过分离传输，可以允许全速/低速设备连接到高速主机。分离传输对于 USB 设备来说是透明的、不可见的。

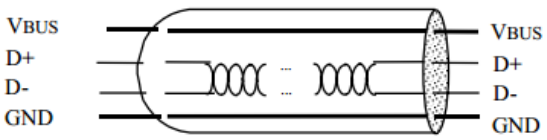
分离传输，顾名思义就是把一次完整的事务传输分成两个事务传输来完成。其出发点是高速传输和全速/低速传输的速度不相等，如果使用一次完整的事务来传输，势必会造成比较长的等待时间，从而降低了高速 USB 总线的利用率。通过将一次传输分成两次，将令牌(和数据)的传输与响应数据(和握手)的传输分开，这样就可以在中间插入其他高速传输，从而提高总线的利用率。

3 USB 物理规范和电气规范

该部分主要针对 USB 2.0 规范的第六章和第七章。

3.1 USB 线缆

USB 规范要求高速和全速 USB 线缆必须使用内含双绞线的屏蔽线。低速 USB 线缆推荐但不要求使用屏蔽双绞线。一条 USB 传输线分别由地线、电源线、D+和 D-四条线构成，D+和 D-是差分输入线。它使用的是 3.3V 的电压（与 CMOS 的 5V 电平不同），而电源线和地线可向设备提供 5V 电压，最大电流为 500mA (可以在编程中设置)。USB 支持“总线供电”和“自供电”两种供电模式。在总线供电模式下，设备最多可以获得 500mA 的电流。



3.2 电气规范

USB2.0 支持 3 种传输速度：低速模式（1.5Mb/s）、全速模式（12Mb/s）和高速模式（480Mb/s）。在低速和全速模式中，采用的是电压传输模式；高速模式下，是电流传输模式。

USB2.0 规范要求集线器必须支持高速模式，USB2.0 设备则没有这个要求。一个高速的上行收发器不能支持低速信号传输模式。而 USB2.0 下行收发器必须支持高速、全速和低速模式。

3.2.1 USB 的插入检测机制

在 USB 集线器的每个下游端口的 D+和 D-上，分别接了一个 15kΩ的下拉电阻。当集线器端口没有设备插入时，输入端被这两个下拉电阻拉到了低电平。而在 USB 设备端，在 D+或者 D-上，接了一个 1.5kΩ的上拉电阻到 3.3V 的电源。对于高速设备和全速设备，上拉电阻接在 D+上；低速设备，上拉电阻接在 D-上。

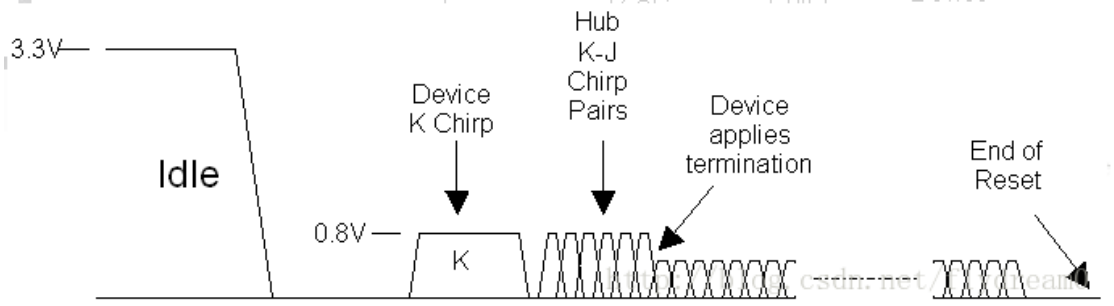
当设备插入到集线器时，接了上拉电阻的数据线的电压由 1.5kΩ的上拉电阻和 15kΩ的下拉电阻分压，大概 3V 左右。这是一个高电平信号。集线器检测到该状态后就上报给 Host 控制器，这样就检测到设备插入。通过检测高电平的数据线是 D+还是 D-来判断设备是什么速度类型。

高速设备会首先被判断为全速设备，然后通过集线器和设备两者确认，切换到高速模式下。高速模式下是电流传输模式，此时需要把 D+的上拉电阻断开。

3.2.2 高速设备握手识别

高速设备会首先被判断为全速设备。之后，hub 和设备通过一系列握手信号确认双方的身份。这里对速度的检测是双向的，比如高速的 hub 需要检测所插入设备是高速、全速还是低速，高速的设备需要检测所连上的 hub 是 USB2.0 的还是 1.x 的，如果是前者，就进行一系列动作切到高速模式工作，否则就以全速模式工作。

下图为高速设备连到 USB2.0 hub 上的协商过程示意图（图片来自网络）。hub 检测到有设备插入，向主机发送 Set_Port_Feature 请求让 hub 复位新插入的设备。设备复位操作是 hub 通过驱动数据线到复位状态 SE0 并持续至少 10ms（Reset 信号）。



高速设备看到复位信号后，通过内部的电流源向 D-线持续灌大小为 17.78mA 电流。因为此时高速设备的 1.5k 上拉电阻还未撤销，在 hub 端，全速/低速驱动器形成一个阻抗为 45 欧姆(Ohm)的终端电阻，2 电阻并联后仍是 45 欧姆左右的阻抗，所以在 hub 端看到一个约 800mV 的电压（45 欧姆*17.78mA），这就是 Chirp K 信号。Chirp K 信号的持续时间是 1ms~7ms。

在 hub 端，虽然下达了复位信号，并一直驱动着 SE0，但 USB2.0 的高速接收器一直在检测 Chirp K 信号，如果没有 Chirp K 信号看到，就继续复位操作，直到复位结束，之后就在全速模式下操作。如果只是一个全速的 hub，不支持高速操作，那么该 hub 不理会设备发送的 Chirp K 信号，之后设备也不会切换到高速模式。

设备发送的 Chirp K 信号结束后 100us 内，hub 必须开始回复一连串的 KJKJKJ…序列，向设备表明这是一个 USB2.0 的 hub。这里的 KJ 序列是连续的，中间不能间断，而且每个 K 或 J 的持续时间在 40us~60us 之间。KJ 序列停止后的 100~500us 内结束复位操作。hub 发送 Chirp KJ 序列的方式和设备一样，通过电流源向差分数据线交替灌 17.78mA 的电流实现。

再回到设备端来。设备检测到 hub 发出的 6 个 Chirp 信号后（3 对 KJ 序列），它必须在 500us 内切换到高速模式。切换动作有：

1. 断开 1.5k 的上拉电阻。

2. 连接 D+/D-上的高速终端电阻（high-speed termination），实际上就是全速/低速差分驱动器。

3. 进入默认的高速状态。

执行 1，2 两步后，USB 信号线上看到的现象就发生变化了：hub 发送出来的 Chirp KJ 序列幅值降到了原先的一半，400mV。这是因为设备端挂载新的终端电阻后，配上原先 hub 端的终端电阻，并联后的阻抗是 22.5 欧姆。400mV 就是由 17.78mA*22.5Ohm 得来。以后高速操作的信号幅值就是 400mV 而不像全速/低速那样的 3V。

至此，高速设备与 USB2.0 hub 握手完毕，进行后续的 480Mbps 高速信号通信。

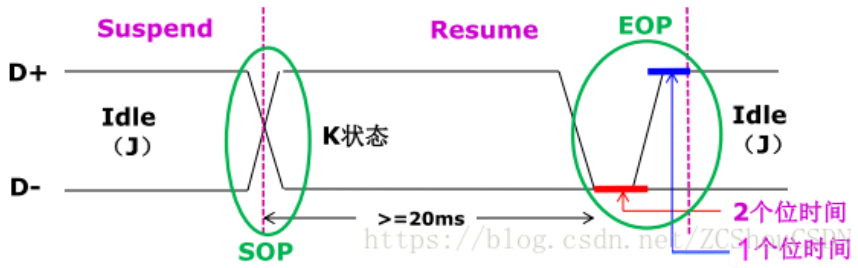
3.2.3 信号

在 USB2.0 规范中，定义了以下一些电平信号：

- 差分信号 1：D+>2.8V，D-<0.3V；
- 差分信号 0：D+<0.3V，D->2.8V。

USB 设备的一些重要信号

信号	低速	全速	高速
J 态	D+:0，D-:1	D+:1，D-:0	D+:1，D-:0
K 态	D+:1，D-:0	D+:0，D-:1	D+:0，D-:1
SE0 态	D+:0　D-:0，用于通知包结束 EOP		
IDLE	K 态	J 态	SE0 态
SOP	Start-Of-Packet，包开始信号，从 IDLE 状态切换到 K 状态。该电平转换代表了同步域的第一个 bit。		
EOP	End-Of-Packet，SE0 态用来通知包结束（EOP），持续 2 位时间的 SE0 信号，后跟随 1 位时间的 J 状态		
SYNC	3 个 K、J 状态切换，后跟随 2 位时间的 K 状态（看到的波形变化是总线上发送 0000 0001 经过 NRZI 编码后的波形）		



SOP 和 EOP 图示

3.2.3.1 Reset 和 Resume 信号

（1）低速 / 全速模式的 Reset 和 Resume 信号

- **Reset 信号：**SE0 状态保持 10ms。主机在要和设备通信之前发送 Reset 信号来把设备配置到默认的未配置状态。复位信号至少持续 10ms。一个设备如果见其上行端口的 SE0 态超过 2.5us，就把它当成复位信号处理。在清除复位信号 10ms 的复位恢复时间后，Hub 必须能接收所有 Hub 请求，设备也必须接收 SetAddress 请求，接收请求失败将导致设备不能被 USB 系统识别。
- **Resume 信号：**20ms 的 K 状态 + 低速 EOP。带远程唤醒功能的设备还可自己发起

唤醒信号；前提是设备已进入 idle 状态至少 5ms，然后发出唤醒 K 信号，维持 1ms 到 15ms 并由主机在 1ms 内接管来继续驱动唤醒信号。

（2）高速设备的 Reset

先确认是高速 Device；Hub 开始发出 SE0 信号，Device 检测 SE0 信号：

1) 如果是从挂起状态唤醒，则 Device 在检查到 2.5us 的 reset 信号后，启动高速检测握手进程；
2) 如果是从全速的挂起状态唤醒，则 Device 在检查到 2.5us~3ms 的 reset 信号后，启动高速检测握手进程；
3) 如果是从高速的非挂起状态唤醒，则 Device 切换到全速前，必须等待 3ms.125ms，在切换到全速后的 100us~875us 间，如果检测到 SE0 信号，启动高速检测握手进程。

3.2.3.2 挂起（Suspend）

在 USB 系统中，正常状态下 hub 或 root hub 会一直周期性地发送 SOF 包(Start Of Frame，全速 USB 每 1ms 发送一个，高速 USB 则是 125μs 发送一个)。根据 USB 协议，如果 USB 线上一直处于空闲(Idle)状态超过 3ms，设备应该把它当作一个挂起(Suspended)信号，要求设备在 10ms 内进入挂起状态，并把设备所需的电流大小降到规定的值（对于 low-power 设备，要求是 500μA，而对于 high-power 或支持远程唤醒(remote wakeup)功能的设备是 2.5mA）。在挂起状态中，设备必须继续向数据项 D+/D-的上拉电阻提供电压以维持 Idle 状态。而对于 USB2.0 高速设备，还有些额外的要求：

1. 高速设备在收到挂起信号（3ms 空闲）后，应在 0.125ms 内切换到全速状态，也就是说要把终端电阻移除，并在 D+数据线上重新挂上 1.5k 上拉电阻。

2. 设备在随后的 100μs-875μs 内检测数据线上的状态。如果该状态是一个 Full speed J，那么说明 host 发下来的是一个挂起信号；如果此时该状态是 SE0，说明是 host drive 数据线 D+到 0，这是一个**复位**信号（复位信号会持续至少 10ms 时间）。

注意：高速设备在挂起状态时处于高速模式，只是所用的是全速信号。一旦从挂起状态恢复回来，会马上进入高速工作状态而无需进行复位操作。

（1）全局挂起：

总线上任何地方都没有通信需求时，整个总线都进入了挂起状态，即全局挂起。Host 通过停止它的所有发射（包括 SOF）来通知全局挂起的开始。尚未挂起的下游设备会在其上游端口上检测到空闲状态，然后自行进入挂起状态。当总线上的设备检测到总线在相当长

的一段时间都处于空闲状态，它便进入挂起状态。Windows 不会以这种方式实现全局挂起。Windows 在停止总线上的所有 USB 通信前会将每个 USB 设备选择挂起（Selective Suspending）。

（2）选择挂起：Selective Suspend

通过发送 SetPortFeature(PORT_SUSPEND)命令到连接了该设备的集线器端口上，使得总线上的该部分可被选择挂起。该端口将阻塞挂起的总线部分，设备将延时一段时间后进入挂起状态。

USB 选择性挂起功能允许集线器驱动程序挂起单个端口，而不会影响集线器上其他端口的操作。USB 设备的选择性挂起在便携式计算机中有助于节省电池电量。许多设备（例如指纹读取器和其他种类的生物识别设备）仅间歇性地需要电源。当不使用设备时，将其挂起可降低总体功耗。

3.2.3.3 唤醒（Resume）

若设备处于挂起状态，当它的上行端口接收到了任何非空闲信号传输，它将被唤醒。此外，假如设备的远程唤醒功能被 USB 系统使能，它也可以发信号给系统以唤醒操作。

主机可以在任何发出唤醒信号。它必须发送至少持续 **20ms** 的唤醒信号，然后以两种方法中的一种来结束唤醒信号，具体方法取决于端口在挂起时的工作速度。若挂起端口处于低速/全速模式，唤醒信号必须以一个标准的低速 EOP 来结束；若挂起端口处于高速模式，唤醒信号必须以一个转到高速空闲状态的转变来结束。

带远程唤醒功能的设备，可能并不会发出唤醒信号，除非总线已经持续空闲状态 5ms。远程唤醒设备必须保持唤醒信号至少 1ms，但不要超过 5ms。在这段时间的结束时刻，设备停止驱动总线。

USB 系统软件在它不打算尝试接收任何连接到刚唤醒的总线部分上的设备期间，必须提供一个 10ms 的唤醒恢复时间。

端口的连接与断开也会导致集线器发送唤醒信号，并唤醒系统。这些事件仅在集线器已被远程唤醒源使能的情况下，导致集线器发送唤醒信号。

3.2.3.4 附图

Table 7-2. Low-/full-speed Signaling Levels

Bus State	Signaling Levels		
	At originating source connector (at end of bit time)	At final target connector	
		Required	Acceptable
Differential "1"	D+ > V_{OH} (min) and D- < V_{OL} (max)	(D+) - (D-) > 200 mV and D+ > V_{IH} (min)	(D+) - (D-) > 200 mV
Differential "0"	D- > V_{OH} (min) and D+ < V_{OL} (max)	(D-) - (D+) > 200 mV and D- > V_{IH} (min)	(D-) - (D+) > 200 mV
Single-ended 0 (SE0)	D+ and D- < V_{OL} (max)	D+ and D- < V_{IL} (max)	D+ and D- < V_{IH} (min)
Single-ended 1 (SE1)	D+ and D- > V_{OSE1}(min)	D+ and D- > V_{IL} (max)	
Data J state: Low-speed Full-speed	 Differential "0" Differential "1"	 Differential "0" Differential "1"	
Data K state: Low-speed Full-speed	 Differential "1" Differential "0"	 Differential "1" Differential "0"	
Idle state: Low-speed Full-speed	NA	 D- > V_{IHZ} (min) and D+ < V_{IL} (max) D+ > V_{IHZ} (min) and D- < V_{IL} (max)	 D- > V_{IHZ} (min) and D+ < V_{IH} (min) D+ > V_{IHZ} (min) and D- < V_{IH} (min)
Resume state	Data K state	Data K state	
Start-of-Packet (SOP)	Data lines switch from Idle to K state		
End-of-Packet (EOP)[4]	SE0 for approximately 2 bit times[1] followed by a J for 1 bit time[3]	SE0 for ≥ 1 bit time[2] followed by a J state for 1 bit time	SE0 for ≥ 1 bit time[2] followed by a J state
Disconnect (at downstream port)	NA	SE0 for ≥2.5 µs	
Connect (at downstream port)	NA	Idle for ≥2 ms	Idle for ≥2.5 µs
Reset	D+ and D- < VOL (max) for ≥10ms	D+ and D- < VIL (max) for ≥10 ms	D+ and D- < VIL (max) for ≥2.5 µs

Table 7-3. High-speed Signaling Levels

Bus State	Required Signaling Level at Source Connector	Required Signaling Level at Target Connector
High-speed Differential "1"	**DC Levels:** V_{HSOH} (min) ≤ D+ ≤ V_{HSOH} (max) V_{HSOL} (min) ≤ D- ≤ V_{HSOL} (max) See Note 1. **AC Differential Levels:** A transmitter must conform to the eye pattern templates called out in Section 7.1.2. See Note 2.	**AC Differential Levels** The signal at the target connector must be recoverable, as defined by the eye pattern templates called out in Section 7.1.2. See Note 2.
High-speed Differential "0"	**DC Levels:** V_{HSOH} (min) ≤ D- ≤ V_{HSOH} (max) V_{HSOL} (min) ≤ D+ ≤ V_{HSOL} (max) See Note 1. **AC Differential Levels:** A transmitter must conform to the eye pattern templates called out in Section 7.1.2. See Note 2.	**AC Differential Levels:** The signal at the target connector must be recoverable, as defined by the eye pattern templates called out in Section 7.1.2. See Note 2.
High-speed J State	High-speed Differential "1"	High-speed Differential "1"
High-speed K State	High-speed Differential "0"	High-speed Differential "0"

Table 7-3. High-speed Signaling Levels (Continued)

Chirp J State (differential voltage; applies only during reset when both hub and device are high-speed capable)	**DC Levels:** V_{CHIRPJ} (min) ≤ (D+ - D-) ≤ V_{CHIRPJ} (max)	**AC Differential Levels** The differential signal at the target connector must be ≥ 300 mV
Chirp K State (differential voltage; applies only during reset when both hub and device are high-speed capable)	**DC Levels:** V_{CHIRPK} (min) ≤ (D+ - D-) ≤ V_{CHIRPK} (max)	**AC Differential Levels** The differential signal at the target connector must be ≤ -300 mV
High-speed Squelch State	NA	V_{HSSQ} - Receiver must indicate squelch when magnitude of differential voltage is ≤100 mV; receiver must not indicate squelch if magnitude of differential voltage is ≥150 mV. See Note 3.
High-speed Idle State	NA	**DC Levels:** V_{HSOI} min ≤ (D+, D-) ≤ V_{HSOI} max See Note 1. **AC Differential Levels:** Magnitude of differential voltage is ≤ 100 mV See Note 3.
Start of High-speed Packet (HSSOP)	Data lines switch from high-speed Idle to high-speed J or high-speed K state.	
End of High-speed Packet (HSEOP)	Data lines switch from high-speed J or K to high-speed Idle state.	
High-speed Disconnect State (at downstream facing port)	NA	V_{HSDSC} - Downstream facing port must not indicate device disconnection if differential voltage is ≤ 525 mV, and must indicate device disconnection when magnitude of differential voltage is ≥ 625 mV, at the sample time discussed in Section 7.1.7.3.

4 USB 通信协议

4.1 字节顺序

数据在 USB 线里传送是**由低位到高位发送的(LSB 在前，MSB 在后)**。USB 采用 NRZI(非归零编码)对发送的数据包进行编码。即：输入数据 0，编码成“电平翻转”；输入数据 1，编码成“电平保持”，如下图。

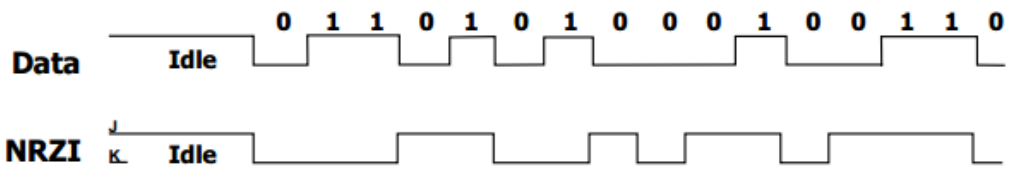


Figure 7-31. NRZI Data Encoding

4.2 USB 数据格式

USB 数据是由二进制数字串构成的，首先数字串组成域（有七种）；**域再组成包；包再组成事务（IN、OUT、SETUP）；事务最后组成传输（中断传输、并行传输、批量传输和控制传输）**。

USB 协议规定的四种传输（transfer）类型中，批量传输、同步传输和中断传输每传输一次数据都是一个事务，控制传输包括三个过程，建立过程和状态过程分别是一个事务，数据过程则可能包含多个事务。

首先介绍几个关键字：

SOF：Start-of-(micro)Frame，是一个特殊的包。开始帧（SOF）数据包由主机以全速总线每 1.00ms±0.0005ms 和高速总线 125us±0.0625us 的标称速率发出。SOF 数据包由一个 PID 指示数据包类型，后面跟着一个 11 位帧数字段，如下图所示。SOF 令牌包括仅用于令牌的事务处理，其以对应于每个帧的开始的精确时间间隔来分配 SOF 标记和伴随帧号。 所有高速和全速功能（包括集线器）都会收到 SOF 数据包。SOF 令牌不会导致任何接收函数生成返回数据包；因此，SOF 交付给任何给定的功能不能得到保证。

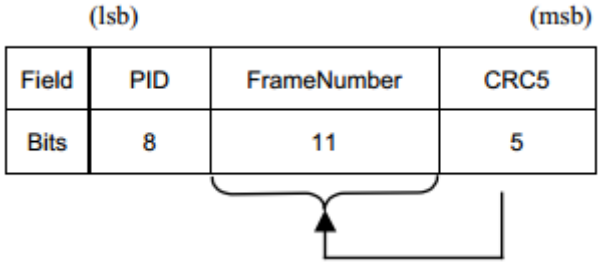


图 3 SOF Packet（Fig8-13）

EOF：End-of-(micro)Frame，是一种电平状态

SOP：Start-of-Packet，定义为从 IDLE 状态切换到 K 状态的电平变化

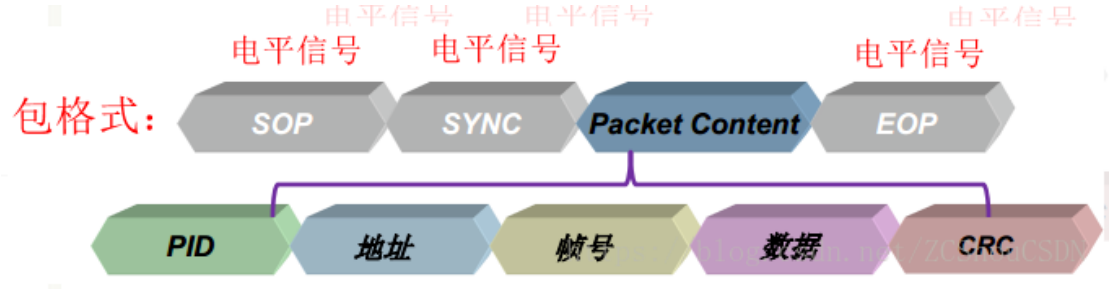
EOP：End-of-Packet，定义为持续 2 位时间的 SE0 信号，后跟随 1 位时间的 J 状态的电平变化。

帧：时间概念，在 USB 中，高速 USB 总线的帧周期为 125us，全速以及低速 USB 总线

的帧周期为 1ms，它是一个独立的单元，包含了一系列总线动作，USB 将 1 帧分为好几份，每一份是一个 USB 的传输动作。帧的起始由一个特定的包（SOF 包）表示，帧尾为 EOF。EOF 不是一个包，而是一种电平状态，EOF 期间不允许有数据传输。

4.2.1 域（Field）

一个包被分为不同域，域是 USB 数据最小的单位，由若干位组成(多少位由具体的域决定)。不同类型的包所包含的域是不一样的，都要**以同步域 SYNC 开始，紧跟一个包标识符 PID，最终 以包结束符 EOP 来结束这个包**。

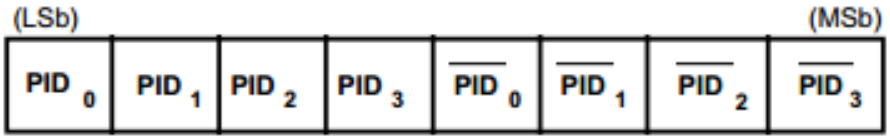


4.2.1.1 同步域(SYNC)

所有的 USB 包都由 SYNC 开始，高速包的 SYNC 宽度为 32bit（31 位 0+1 位 1），全速/低速包的 SYNC 段度为 8bit（7 位 0+1 位 1）。实际接收到的 SYNC 长度由于 USB HUB 的关系，可能会小于该值。同步域用于告诉 USB 串行接口引擎数据即将开始传输，另一个作用是用于本地时钟与输入同步。

4.2.1.2 标识域(PID)

PID 是用来标识一个包的类型的。它共有 8 位，只使用 4 位（PID0 ~ PID3），另外 4 位是 PID0 ~ PID3 的取反，用来校验 PID。PID 规定了四类包：令牌包、数据包、握手包和特殊包。同类的包又各分为具体的四种包。



PID 格式（Fig8-1）

PID Type	PID Name	PID<3:0>*	Description
Token	OUT	0001B	启动一个方向为主机到设备的传输，并包含了设备地址和标号。
	IN	1001B	启动一个方向为设备到主机的传输，并包含了设备地址和标号。
	SOF	0101B	表示一个帧的开始，并且包含了相应的帧号
	SETUP	1101B	启动一个控制传输，用于主机对设备的初始化
Data	DATA0	0011B	偶数据包
	DATA1	1011B	奇数据包
	DATA2	0111B	Data packet PID high-speed, high bandwidth isochronous transaction in a microframe (see Section 5.9.2 for more information)
	MDATA	1111B	Data packet PID high-speed for split and high bandwidth isochronous transactions (see Sections 5.9.2, 11.20, and 11.21 for more information)
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Receiving device cannot accept data or transmitting device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported
	NYET	0110B	No response yet from receiver (see Sections 8.5.1 and 11.17-11.21)
Special	PRE	1100B	(Token) Host-issued preamble. Enables downstream bus traffic to low-speed devices.
	ERR	1100B	(Handshake) Split Transaction Error Handshake (reuses PRE value)
	SPLIT	1000B	(Token) High-speed Split Transaction Token (see Section 8.4.2)
	PING	0100B	(Token) High-speed flow control probe for a bulk/control endpoint (see Section 8.5.1)
	Reserved	0000B	Reserved PID

*Note: PID bits are shown in MSb order. When sent on the USB, the rightmost bit (bit 0) will be sent first.

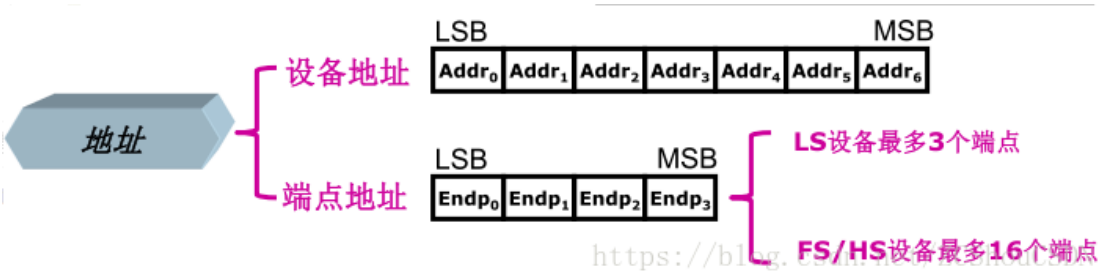
PID 类型（Table8-1）

4.2.1.3 地址域(ADDR)

地址共占 11 位，其中低 7 位是设备地址，高 4 位是端点地址。

a) 地址域：七位地址，代表了设备在主机上的地址，地址 000 0000 被命名为零地址，是任何设备第一次连接到主机时，在被主机配置、枚举前的默认地址，由此可以知道为什么一个 USB 主机只能接 127 个设备。

b) 端点域(ENDP)：四位，由此可知一个 USB 设备端点数量最大为 16 个。



地址域格式

4.2.1.4 帧号

占 11 位，主机每发出一个帧，帧号都会自加 1，当帧号达到 0x7FF 时，将归零重新开始计数。帧号域最大容量 0x800，对于同步传输有重要意义。

4.2.1.5 数据

根据传输类型的不同，数据域的数据长度从 0 到 1024 字节不等。

	Control			Bulk			Interrupt			isoch		
	HS	FS	LS	HS	FS	LS	HS	FS	LS	HS	FS	LS
数据包长度	64	64	8	512	64	N.A	1024	64	8	1024	1023	N.A

数据包长度

4.2.1.6 CRC

对令牌包和数据包中非 PID 域进行校验的一种方法，CRC 校验在通讯中应用很泛，是一种很好的校验方法，CRC 码的除法是模 2 运算，不同于 10 进制中的除法。

a) Token CRCs：对于令牌（Token）使用 **5 位 CRC**，涵盖了 IN，SETUP 和 OUT 令牌的 ADDR 和 ENDP 字段或 SOF 令牌的时间戳字段。 PING 和 SPLIT 特殊令牌也包括一个 5 位 CRC 字段。

b) Data CRCs：数据 CRC 是应用在数据包的数据字段上的 16 位多项式，使用 **16 位 CRC**。

4.2.2 包（Packet）

包（Packet）是 USB 系统中信息传输的基本单元，所有数据都是经过打包后在总线上传输的。**包是 USB 总线上数据传输的最小单位**，不能被打断或干扰，否则会引发错误。若干个数据包组成一次事务传输。一次事务传输也不能打断，属于一次事务传输的几个包必须连续，不能跨帧完成。一次传输由一次到多次事务传输构成，可以跨帧完成。

由域构成的包有四种类型，分别是**令牌包、数据包、握手包和特殊包**，不同包的域结构不同。

4.2.2.1 令牌包

令牌包分为输入包、输出包、设置包和帧起始包（注意这里的输入包是用于设置输入命令的，输出包是用来设置输出命令的，而不是放数据的）其中输入包、输出包和设置包的格式都是一样的：

`SYNC + PID + （ADDR + ENDP） + CRC5(五位的校验码)`

帧起始包的格式：`SYNC + PID + 11 位 FRAM + CRC5(五位的校验码)`

输出（OUT）令牌包用来通知设备将要输出一个数据包；输入（IN）令牌包用来通知设备返回一个数据包；建立（SETUP）令牌包只用在控制传输中，通知设备将要输出一个数据包；帧起始包在每帧（或微帧）开始时以广播的形式发送，所有 USB 全速设备和高速设备都可以接收到 SOF 包。

4.2.2.2 数据包

数据包分为 DATA0 包和 DATA1 包。当 USB 发送数据的时候，如果一次发送的数据长度大于相应端点的容量时，就需要把数据包分为好几个包分批发送，DATA0 包和 DATA1 包交替发送，即如果第一个数据包是 DATA0，那第二个数据包就是 DATA1。**但也有例外情况，在同步传输中(四类传输类型中之一)，所有的数据包都是为 DATA0**。格式如下：

`SYNC + PID + 0~1024 字节 + CRC16`

注意：低速设备允许的最大数据有效载荷大小为 8 个字节。 全速设备的最大数据有效载荷大小为 1023.高速设备的最大数据有效载荷大小为 1024 个字节。

4.2.2.3 握手包

握手包包括 ACK、NAK、STALL 以及 NYET 四种，其中

- ACK 表示肯定的应答成功的数据传输。对于 IN 事务，将由 host 发出；对于 OUT、SETUP 和 PING 事务，将由 device 发出。
- NAK 表示否定的应答失败的数据传输，要求重新传输。在数据阶段，对于 IN 事务，它将由 device 发出；在握手阶段，对于 OUT 和 PING 事务，它也将由 device 发出；host 从不发送 NAK 包。
- STALL 表示功能错误或端点被设置了 STALL 属性。
- NYET 表示尚未准备好，要求等待。

握手包是结构最为简单的包，格式如下：`SYNC + PID`

4.2.3 事务（Transaction）

在 USB 上数据信息的一次接收或发送的处理过程称为事务处理（Transaction），分别有

IN、OUT 和 SETUP 三大事务。一个事务由一系列 packet 组成，具体由哪些 packet 组成，它取决于具体的事务，可能由一个 token packet、可选的 data packet、可选的 handshake packet 和可选的 special packet 组成。

事务包括三种类型：IN、OUT 和 SETUP。

4.2.3.1 IN 事务

表示 USB 主机从总线上的某个 USB 设备接收一个数据包的过程。

- 令牌包阶段——主机发送一个 PID 为 IN 的输入包给设备，通知设备要往主机发送数据；
- 数据包阶段——设备根据情况会作出三种反应(要注意：数据包阶段也不总是传送数据的，根据传输情况还会提前进入握手包阶段)。
- 握手包阶段——主机正确接收到数据之后就会向设备发送 ACK 包。

【正常】的输入事务处理：设备往主机里面发出数据包(DATA0 与 DATA1 交替)

1.主机->设备(令牌信息包)	SYNC	IN	ADDR	ENDP	CRC5
2.设备->主机(数据信息包)	SYNC	DATA0	DATA		CRC16
3.主机->设备(握手信息包)	SYNC	ACK			

【设备忙】时的输入事务处理：无法往主机发出数据包就发送 NAK 无效包，IN 事务提前结束，到了下一个 IN 事务才继续；

1. 主机->设备(令牌信息包)	SYNC	IN	ADDR	ENDP	CRC5
2. 设备->主机(握手信息包)	SYNC	NAK			

【设备出错】时的输入事务处理：发送错误包 STALL 包，事务也就提前结束了，总线进入空闲状态

1. 主机->设备(令牌信息包)	SYNC	IN	ADDR	ENDP	CRC5
2. 设备->主机(握手信息包)	SYNC	STALL			

4.2.3.2 OUT 事务

输出事务处理：表示 USB 主机把一个数据包输出到总线上的某个 USB 设备接收的过程。

- 令牌包阶段——主机发送一个 PID 为 OUT 的输出包给设备，通知设备要接收数据；
- 数据包阶段——比较简单，就是主机会往设备送数据，DATA0 与 DATA1 交替
- 握手包阶段——设备根据情况会作出三种反应

【正常】的输出事务处理：设备给主机返回 ACK，通知主机可以发送新的数据，如果数据包发生了 CRC 校验错误，将不返回任何握手信息；

1. 主机->设备(令牌信息包)	SYNC	OUT	ADDR ENDP	CRC5
2. 主机->设备(数据信息包)	SYNC	DATA0	DATA	CRC16
3. 设备->主机(握手信息包)	SYNC	ACK		

【设备忙时】的输出事务处理：无法给主机返回 ACK，就发送 NAK 无效包，通知主机再次发送数据

1. 主机->设备(令牌信息包)	SYNC	OUT	ADDR	ENDP	CRC5
2. 主机->设备(数据信息包)	SYNC	DATA0	DATA		CRC16
3. 设备->主机(握手信息包)	SYNC	NAK			

【设备出错】的输出事务处理：发送错误包 STALL 包，事务提前结束，总线进入空闲状态。

1. 主机->设备(令牌信息包)	SYNC	OUT	ADDR	ENDP	CRC5
2. 主机->设备(数据信息包)	SYNC	DATA0	DATA		CRC16
3. 设备->主机(握手信息包)	SYNC	STALL			

4.2.3.3 SETUP 事务

- 令牌包阶段——主机发送一个 PID 为 SETUP 的输出包给设备，通知设备要接收数据；
- 数据包阶段——比较简单，就是主机往设备送数据，注意，这里只有一个固定为 8 个字节的 **DATA0** 包，这 8 个字节的内容就是标准的 USB 设备请求命令。
- 握手包阶段——设备接收到主机的命令信息后，返回 ACK，此后总线进入空闲状态，并准备下一个传输(在 SETUP 事务后通常是一个 IN 或 OUT 事务构成的传输)。

【正常】的设置事务处理

1. 主机->设备(令牌信息包)	SYNC	SETUP	ADDR	ENDP	CRC5
2. 主机->设备(数据信息包)	SYNC	DATA0	DATA		CRC16
3. 设备->主机(握手信息包)	SYNC	ACK			

【设备忙时】的设置事务处理

1. 主机->设备(令牌信息包)	SYNC	SETUP	ADDR	ENDP	CRC5
2. 主机->设备(数据信息包)	SYNC	DATA0	DATA		CRC16
3. 设备->主机(握手信息包)	SYNC	NAK			

【设备出错】的设置事务处理

1. 主机->设备(令牌信息包)	SYNC	SETUP	ADDR	ENDP	CRC5
2. 主机->设备(数据信息包)	SYNC	DATA0	DATA		CRC16
3. 设备->主机(握手信息包)	SYNC	STALL			

5 USB Device Framework

本章对应 USB2.0 协议第 9 章。

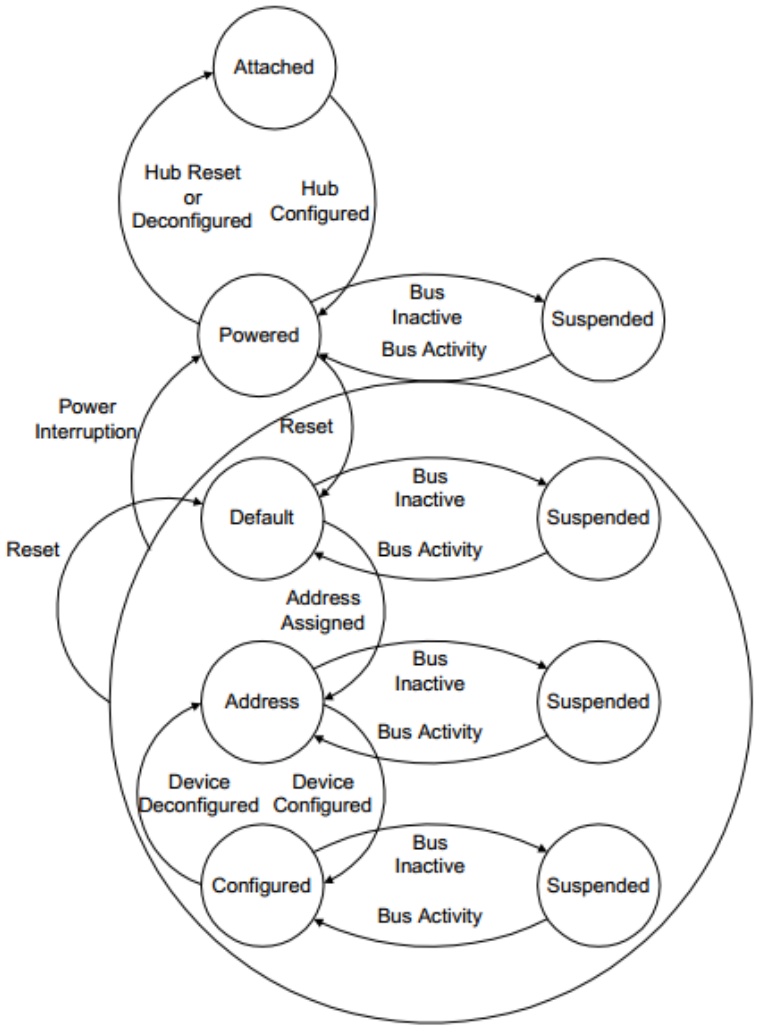
一个 USB 设备可以划分为三层：

- 底层：负责发送和接收数据的总线接口
- 中间层：处理总线接口和设备端点之间的数据
- 上层：功能层

本章主要论述中间层。

5.1 USB 设备状态

USB 设备的 6 种状态：连接状态、上电状态、默认状态、地址状态、配置状态、挂起状态，六者之间的关系图如下所示：



5.1.1 连接(Attached)

设备可以连接到 USB 或者从 USB 上拔出。USB 设备从总线上拔出后的状态在规范没

定义，只说明一旦 USB 连到总线要求的操作以及属性。

5.1.2 上电(Powered)

设备可以支持自供电和总线供电配置。 某些设备配置支持任意电源。 仅当设备是自供电时，其他设备配置才可用。**设备通过配置描述符（配置描述符见 5.6.3）报告其电源功能**。当前电源被报告为设备状态的一部分。设备可以在任何时间改变其电源，例如从自供电到总线供电。如果配置能够支持两种功耗模式，则为该配置报告的功率最大值是设备在任一模式下从 VBUS 中获取的最大值。无论模式如何，设备都必须遵守此最大值。如果配置仅支持一种电源模式且设备的电源发生变化，则设备将丢失其当前配置和地址并返回到 Powered 状态。如果设备是自供电的并且其当前配置需要超过 100 mA，那么如果设备切换为总线供电，则必须返回到地址状态。如果本地电源丢失，则允许使用 VBUS 为集线器控制器供电的自供电集线器保持在已配置状态。

USB 设备的电源可来自外部电源，也可从 USB 接口的集线器而来。电源来自外部电源的 USB 设备被称作自给电源式的(self-powered)。尽管自给电源式的 USB 设备可能在连接上 USB 接口以前可能已经带电，但它们直到连线上 USB 接口后才能被看作是加电状态(Powered state)。而这时候 VBUS 已经对设备产生作用了。

为检测 Hub 端口设备拔插状态，Hub 的端口必须供电。总线供电的 Hub 在其端口没有被配置之前不必向下游端口供电，

5.1.3 默认状态(Default)

设备上电后，它不响应任何总线处理，直到总线接收到复位信号为止。接收到复位信号后，用默认的地址可以对设备寻址。

当用复位过程完成后，USB 设备在正确的速度下操作(即低速/全速/高速)。低速和全速的数据选择由设备的终端电阻决定。能进行高速操作的设备决定它是否在复位的过程的一部分执行高速操作。

能进行高速操作的设备在全速的电气环境中操作时，必须能以全速成功复位。设备成功复位后，设备必须成功响应设备和配置描述符请求，并且返回适当的信息。当在全速下工作时,设备可能或者不能支持预定义的功能。

5.1.4 地址(Address)

所有的 USB 设备在加电复位以后都使用缺省地址。每一设备在连接或复位后由主机分配一个唯一的地址。当 USB 设备处于挂起状态时，它保持这个地址不变。USB 设备只对默认通道(Pipe)请求发生响应，而不管设备是否已经被分配地址或在使用默认地址。

5.1.5 配置状态(Configured)

在 USB 设备正常工作以前，设备必须被正确配置。从设备的角度来看，配置包括一个将非零值写入设备配置寄存器的操作。配置一个设备或改变一个可变的设备设置会使得与这个相关接口的终端结点的所有的状态与配置值被设成缺省值。这包括将正在使用(date toggle)的结点(end point)的 (Date toggle)被设置成 DATA0。

5.1.6 挂起状态(Suspended)

为节省电源，USB 设备在探测不到总线传输时自动进入挂起状态。当挂起时，USB 设备保持本身的内部状态，包括它的地址及配置。

所有的设备在一段特定的时间内探测不到总线活动时必须进入挂起态。不管设备是被分配了非缺省的地址或者是已被配置，已经连接的设备必须在任何加电的时刻随时准备被挂起。总线活动的中止可能是因为主机本身进入了挂起状态。另外，USB 设备必须在所连接的集线器端口失效时进入中止态。这就是所指的**选择性中止(Selective suspend)**。

USB 设备在总线活动来到时结束挂起态。USB 设备也可以远程唤醒的电流信号来请求主机退出中止态或选择性中止态。具体设备具有的远程唤醒的能力是可选的，也就是说，如果一个设备有远程唤醒的能力，此设备必须能让主机控制此能力的有效与否。当设备复位时，远程唤醒能力必须被禁止。

5.2 总线枚举(Bus Enumeration)

供电——总线复位——获取设备描述符——总线复位——设置地址——获取设备描述符——获取配置描述符——获取字符串描述符——…设置配置…

USB 设备连接到主机或从主机上拔出时，主机通过枚举(Enumeration)过程来识别和管理设备。当一个设备连接到一个供电的端口时，将会有下列过程：

1.检测电压变化，报告主机：USB 设备上电后，HUB 检测到有电压变化，将利用自己的中断端点将信息反馈给主控制器有设备连接。

2.主机了解连接设备：主机在知道有设备接入后会发送一个 Get_Port_Status 请求(request)给 hub 以了解此次状态改变的确切含义。

3.Hub 检测所插入的设备是高速还是低速：hub 通过检测 USB 总线空闲(Idle)时差分线的高低电压来判断所连接设备的速度类型，当 host 发来 Get_Port_Status 请求时，hub 就可以将此设备的速度类型信息回复给 host。USB 2.0 规范要求速度检测要先于复位（Reset）操作。

4.hub 复位设备：主机一旦得知新设备已连上以后，它至少等待 100ms 以使得插入操作的完成以及设备电源稳定工作。然后主机控制器就向 hub 发出一个 Set_Port_Feature 请求让 hub 复位其管理的端口(刚才设备插上的端口)。hub 通过驱动数据线到复位状态(D+和 D-全为低电平)，并持续至少 10ms。当然，hub 不会把这样的复位信号发送给其他已有设备连接的端口，所以其他连在该 hub 上的设备自然看不到复位信号，不受影响。

5.Host 检测所连接的全速设备是否是支持高速模式：根据 USB 2.0 协议，高速（High Speed）设备在初始时是默认全速（Full Speed ）状态运行，所以对于一个支持 USB 2.0 的高速 hub，当它发现它的端口连接的是一个全速设备时，会进行高速检测，看看目前这个设备是否还支持高速传输，如果是，那就切到高速信号模式，否则就一直在全速状态下工作。

6.Hub 建立设备和主机之间的信息通道：主机不停地向 hub 发送 Get_Port_Status 请求，以查询设备是否复位成功。Hub 返回的报告信息中有专门的一位用来标志设备的复位状态。当 hub 撤销了复位信号，设备就处于默认/空闲状态（Default state），准备接收主机发来的请求。设备和主机之间的通信通过**控制传输，默认地址 0，端点号 0** 进行。此时，设备能从总线上得到的最大电流是 100mA。

7.主机发送 Get_Descriptor 请求，获取设备描述符：默认管道（Default Pipe）在设备一端来看就是端点 0。主机此时发送的请求是默认地址 0，端点 0。设备描述符的第 8 字节代表设备端点 0 的最大包大小。当完成第一次的控制传输后，系统会要求 hub 对设备进行再一次的复位操作。再次复位的目的是使设备进入一个确定的状态。

8.主机给设备分配地址：主机控制器通过 Set_Address 请求向设备分配一个唯一的地址。此后，设备进入地址状态（Address state），之后就启用新地址继续与主机通信。

9.主机获取设备的信息：主机再次发送 Get_Descriptor 请求到新地址读取设备描述符。设备描述符内信息包括端点 0 的最大包长度，设备所支持的配置（Configuration）个数，设备类型，VID、PID 等信息。

之后主机发送 Get_Descriptor 请求，读取配置描述符（Configuration Descriptor），字符串描述符等，逐一了解设备更详细的信息。配置描述符总共为 9 字节。主机在获取到配置描述符后，根据里面的配置集合总长度，再获取配置集合。配置集合包括配置描述符，接口描述符，端点描述符等。如果有字符串描述符，还要获取字符串描述符。另外 HID 设备还有 HID 描述符等。

10.主机给设备挂载驱动（复合设备除外）：主机通过解析描述符后对设备有足够的了解，会选择一个最合适的驱动给设备。 然后调用设备模型提供的接口 device_add 将设备添加到

usb 总线的设备列表里，然后 usb 总线会遍历驱动列表里的每个驱动，调用自己的 match(usb_device_match)函数查询驱动和连入的设备或接口是否匹配，匹配的话调用 device_bind_driver 函数，将控制权交到设备驱动。

对于复合设备，通常应该是不同的接口（Interface）配置给不同的驱动，因此，需要等到当设备被配置并把接口使能后才可以把驱动挂载上去。

11.设备驱动选择一个配置：驱动根据前面设备回复的信息，发送 Set_Configuration 请求来正式确定选择设备的哪个配置（Configuration）作为工作配置。至此，设备处于配置状态(Configured)，当然，设备也应该使能它的各个接口（Interface）。

5.3 USB 请求处理的限制

USB 规定任何请求必须在 5s 内处理，实际上 5s 并不切合实际（太长了）。确切的限定时间在以下具体描述。

Reset/Resume 恢复时间：10ms，在一个端口被 Reset 或者 Resume 后的 10ms 后，设备才能够响应数据传输。10ms 内设备不会响应。

Set Address 处理：如果设备接收到 SetAddress()的请求，设备必须在 50ms 内完成该过程和状态阶段（ACK）。状态阶段完成后，USB 允许设备有 2ms 的时间来完成地址的恢复工作。此后，设备能够通过新地址接收 Setup 包。

标准设备请求：对于没有数据阶段的标准设备请求，设备必须能够在 50ms 内完成请求和状态阶段。对于有数据阶段的标准设备请求，设备必须能够在 500ms 内返回第一个数据包给 Host。接下来的数据包，设备必须在每完成一个数据包传送后的 500ms 内返回。数据阶段后，设备需要在 50ms 内完成状态阶段。

Class-specific 请求：要求一般和标准设备请求一样。

5.4 设备请求

所有 USB 设备都会响应设备默认控制管道上来自主机的请求。这些请求使用控制转移进行。请求和请求的参数在 Setup 数据包中发送到设备。主机负责建立下表中列出的字段值。每个 SETUP 包都有八个字节。

Table 9-2. Format of Setup Data

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

SETUP 格式

bmRequestType：该位图字段标识特定请求的特征。特别是，该字段标识控制传输第二阶段中的数据传输方向。如 wLength 字段为零，则表示方向位的状态将被忽略，表示没有数据阶段。USB 规范定义了所有设备必须支持的一系列标准请求。这些在下一节的表中列举。另外，设备类可以定义额外的请求。设备供应商也可以定义设备支持的请求。

可以将请求指向设备，设备上的接口或设备上的特定端点。该字段还指定了请求的预期收件人。当指定接口或端点时，wIndex 字段标识接口或端点。

bRequest：该字段指定了特定的请求。bmRequestType 字段中的 Type 位修改此字段的含义。本规范仅在位重置为零时指定 bRequest 字段的值，表示标准请求见 4.3.1.1。

wValue：该字段的内容根据要求而有所不同。它用于将参数传递给设备，特定于请求。具体见后文的说明。

wIndex：该字段的内容根据要求而有所不同。它用于将参数传递给设备，特定于请求。wIndex 字段通常用于指定端点或接口的请求中。

5.4.1.1 Standard Request Codes

Table 9-4. Standard Request Codes

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
Reserved for future use	2
SET_FEATURE	3
Reserved for future use	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

5.4.1.2 Descriptor Type

Table 9-5. Descriptor Types

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5
DEVICE_QUALIFIER	6
OTHER_SPEED_CONFIGURATION	7
INTERFACE_POWER[1]	8

5.5 标准设备请求

本节介绍为所有 USB 设备定义的标准设备请求。表 9-3 给出了标准设备请求，而表 9-4

和表 9-5 分别给出了标准请求代码和描述符类型。USB 设备必须响应标准的设备请求，即使设备尚未分配地址或尚未配置。Feature selectors 用于启用或设置特定功能，如设备，接口或端点特有的远程唤醒功能。

Table 9-3. Standard Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Table 9-6. Standard Feature Selectors

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	Device	1
ENDPOINT_HALT	Endpoint	0
TEST_MODE	Device	2

5.5.1 Clear Feature

此请求用于清除或禁用特定功能。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

wValue 中的功能选择器值必须适合接收者。当接收者是设备时，只有设备功能选择器值可以使用；当接收者是接口时，只有接口功能选择器值可以使用；并且当接收者是端点时可以使用端点功能选择器值。不存在或引用不存在的接口或端点的功能的 ClearFeature()请求将导致设备响应请求错误。如果 wLength 不为零，则未指定设备行为！

Default state：当设备处于默认状态时收到该请求，则设备行为是未指定的。

Address state：当设备处于地址状态时，此请求有效；对接口或端点零以外的端点的引用会造成设备响应请求错误。

Configured state：当设备处于 Configured 状态时，此请求有效。

5.5.2 Get Configuration

该请求返回当前设备配置值。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value

如果返回值为零，则说明设备未配置。如果 wValue，wIndex 或 wLength 不符合上述规定，则未指定设备行为。

Default state：当设备处于默认状态时收到该请求，则设备行为是未指定的。

Address state：必须返回 0.

Configured state：必须返回当前配置的非零值 bConfigurationValue。

5.5.3 Get Descriptor

如果描述符存在，该请求将返回指定的描述符。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B (0x80)	GET_DESCRIPTOR (0x06)	Low-Byte: Index High-byte: Descriptor Type	0 或语言 ID	描述符长度	描述符

wValue 字段（2 个字节）的高字节中为描述符类型（参见 5.6），低字节指定描述符索引。描述符索引用于在设备中实现多个相同类型的描述符时选择特定描述符（描述符索引仅用于

配置描述符和字符串描述符）。例如，一个设备可以实现多个配置描述符。而对于可以通过GetDescriptor()请求检索其他的标准描述符（例如设备描述符），描述符索引必须使用零。用于描述符索引的值的范围从 0 到比设备实现的那种描述符的数量少一个。

wIndex 字段（2 字节）指定字符串描述符的语言 ID，当不是字符串描述符时，则该值为零。

wLength 字段（2 个字节）指定要返回的字节数。如果描述符长于 wLength 字段，则只返回描述符的初始字节。如果描述符比 wLength 字段短，则当请求进一步的数据时，设备通过发送短分组来指示控制传输的结束。短数据包定义为比最大有效负载大小或长度为零的数据包短的数据包（具体请参阅规范的第 5 章）。

该请求只用于请求以下三种类型的描述符：设备描述符（包括设备限定符device_qualifier）、配置描述符（包括 other_speed_configuration）、字符串描述符。对配置描述符的请求返回单个请求中所有接口的配置描述符，所有接口描述符和端点描述符。第一个接口描述符在配置描述符之后。第一个接口的端点描述符紧随第一个接口描述符。如果有其他接口，它们的接口描述符和端点描述符将紧随第一个接口的端点描述符。

所有设备都必须提供设备描述符和至少一个配置描述符。 如果一个设备不支持请求的描述符，它会回应一个请求错误。

Default state: 此时该请求是有效的。

Address state: 此时该请求是有效的。

Configured state: 此时该请求是有效的。

举例：

1.获取设备描述符报文如下（注意字节顺序（**USB 规定低字节在前**））：

80 06 00 01 00 00 12 00

该报文表示，主机请求设备返回一个描述符（0x06），描述符类型为设备描述符（0x01），长度为 18 字节（0x12）.

2.获取配置描述符的报文如下：

80 06 00 02 00 00 09 00

该报文表示，主机请求设备返回一个描述符（0x06），描述符类型为配置描述符（0x02），长度为 9 字节（0x09）.

3.获取指定索引字符串描述符的报文如下：

80 06 01 03 00 00 09 00

该报文表示，主机请求设备返回一个描述符（0x06），描述符类型为字符串描述符（0x03），字符串是索引是 0x01，长度为 9 字节（0x09）。设备需要返回在设备描述符里第 14-16 个字节索引值设置为 0x01 对应的字符串描述符。

5.5.4 Get Interface

该请求返回指定接口的选定备用设置。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Setting

某些 USB 设备的配置具有互斥设置的接口。该请求允许主机确定当前选择的备用设置。如果 wValue 或 wLength 不符合上述规定，则设备行为未指定。如果指定的接口不存在，那么设备将响应请求错误。

Default state：当设备处于默认状态时收到该请求，则设备行为是未指定的。

Address state：设备给出请求错误响应

Configured state：请求是有效的。

5.5.5 Get Status

此请求返回指定接收者的状态。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status

bmRequestType 字段的 Recipient bits 指定了所需的接收者。返回的数据是指定接收者的当前状态。如果 wValue 或 wLength 不符合上面的规定，或者如果 wIndex 对于设备状态请求非零，则未指定设备的行为。如果指定的接口或端点不存在，那么设备将响应请求错误。

Default state：当设备处于默认状态时收到该请求，则设备行为是未指定的。

Address state：如果指定了端点 0 以外的端点或者是接口，则设备会响应请求错误。

Configured state：如果指定的端点或者接口不存在，设备会响应请求错误。

5.5.5.1 GetStatus()对设备的请求的返回结果格式

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)						Remote Wakeup	Self Powered
D15	D14	D13	D12	D11	D10	D9	D8
Reserved (Reset to zero)							

Figure 9-4. Information Returned by a GetStatus() Request to a Device

Self Powered 字段指示设备当前是否自供电。如果 D0 重置为零，则该设备由总线供电。如果 D0 置为 1，则设备是自供电的。SelfFeature 字段可能不会被 SetFeature()或 ClearFeature()请求更改。

Remote Wakeup 字段指示设备当前是否被允许请求远程唤醒。支持远程唤醒的设备的默认模式被禁用。如果 D1 被重置为零，则设备发信号通知远程唤醒的能力被禁用。如果 D1 设置为 1，则启用设备发送远程唤醒信号的能力。远程唤醒字段可以通过使用 DEVICE_REMOTE_WAKEUP 功能选择器的 SetFeature()和 ClearFeature()请求来修改。

5.5.5.2 GetStatus()对接口的请求的返回结果格式

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							
D15	D14	D13	D12	D11	D10	D9	D8
Reserved (Reset to zero)							

Figure 9-5. Information Returned by a GetStatus() Request to an Interface

5.5.5.3 GetStatus()对端点的请求的返回结果格式

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							Halt
D15	D14	D13	D12	D11	D10	D9	D8
Reserved (Reset to zero)							

Figure 9-6. Information Returned by a GetStatus() Request to an Endpoint

需要为所有中断和批量端点类型实施暂停功能。如果端点当前暂停，则暂停功能设置为 1。否则，暂停功能重置为零。可以选择使用 SetFeature（ENDPOINT_HALT）请求来设置暂停功能。

5.5.6 Set Address

该请求为即将连入的设备设置地址。

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B (0x00)	SET_ADDRESS (0x05)	Device Address	0x00	0x00	None

请求实际上可能会导致最多三个阶段。在第一阶段，设置数据包被发送到设备。在可选的第二阶段，数据在主机和设备之间传输。在最后阶段，状态在主机和设备之间传输。数据和状态传输的方向取决于主机是向设备发送数据还是设备正在向主机发送数据。状态阶段转移始终与数据阶段相反。如果没有数据阶段，则状态阶段从设备到主机。

最初的 SETUP 数据包之后的阶段采用与安装数据包相同的设备地址。直到此请求的状态阶段成功完成后，USB 设备才会更改其设备地址。请注意，这是此请求和所有其他请求之间的区别。对于所有其他请求，指示的操作必须在状态阶段之前完成。

如果指定的设备地址大于 127，或者 wIndex 或 wLength 非零，则未指定设备的行为。

Default state：如果指定的地址非零，则设备应进入地址状态；否则，设备保持默认状态（这不是错误状态）。

Address state：如果指定的地址为零，则设备将进入默认状态；否则，设备保持在地址状态，但使用新指定的地址。

Configured state：设备处于 Configured 状态时收到此请求时的设备行为是未指定的。

举例：

1.设置地址的报文如下

00 05 04 00 00 00 00 00

该报文表示，主机向设备发出设置地址请求（0x05），设置的地址为 0x00,0x04

5.5.7 Set Configuration

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B (0x00)	SET_CONFIGURATION (0x09)	配置值	0x00	0x00	None

wValue 字段的低字节指定了所需的配置。该配置值必须为零或匹配配置描述符中的配置值。如果配置值为零，则设备处于其地址状态。wValue 字段的高字节保留。如果 wIndex，wLength 或 wValue 的高字节不为零，则此请求的行为未指定。

Default state：在设备处于默认状态时收到此请求时的设备行为未指定。

Address state：如果指定的配置值为零，则设备保持在地址状态。 如果指定的配置值

与配置描述符中的配置值相匹配，则选择该配置并且设备进入配置状态。 否则，设备会响应请求错误。

Configured state：如果指定的配置值为零，则设备进入地址状态。 如果指定的配置值与配置描述符中的配置值相匹配，则选择该配置并且设备保持配置状态。 否则，设备会响应请求错误。

5.5.8 Set Descriptor

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Language ID (refer to Section 9.6.7) or zero	Descriptor Length	Descriptor

该请求可选，且可用于更新已有的描述符或添加新的描述符。

wValue 字段的高字节为描述符类型（参见 5.4.1.2）和低字节为描述符索引。描述符索引用于在设备中实现多个相同类型的描述符时选择特定的描述符(仅用于配置和字符串描述符)。例如，一个设备可以实现多个配置描述符。对于可以通过 SetDescriptor()请求设置的其他标准描述符，必须使用零描述符索引。用于描述符索引的值的范围从 0 到比设备实现的那种描述符的数量少一个。

wIndex 字段指定字符串描述符的语言 ID，或者为其他描述符重置为零。wLength 字段指定要从主机传输到设备的字节数。

描述符类型唯一允许的值是设备，配置和字符串描述符类型。

如果此请求不受支持，则设备将响应请求错误。

Default state：在设备处于默认状态时收到此请求时的设备行为未指定。

Address state：如果支持，当设备处于地址状态时，这是一个有效的请求。

Configured state：如果支持，当设备处于 Configured 状态时，这是一个有效的请求。

5.5.9 Set Feature

bmRequestType	bRequest	wValue	wIndex		wLength	Data
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Test Selector	Zero Interface Endpoint	Zero	None

该请求用于使能指定的 feature。

wValue 中的功能选择器值必须适合接收者。当接收者是设备时，只能使用设备功能选择器值；当接收者是接口时，只有接口特征选择器值可以被使用，并且当接收者是端点时，

可以仅使用端点特征选择器值。

引用无法设置或不存在的功能的 SetFeature()请求会导致在请求的状态阶段返回 STALL。

如果 wLength 非零，那么设备的行为没有被指定。

如果指定了不存在的端点或接口，则设备会响应请求错误。

Default state：处于默认状态时，设备必须能够接受 SetFeature(TEST_MODE，TEST_SELECTOR)请求。未指定设备处于默认状态时其他 SetFeature 请求的设备行为。

Address state：如果指定了端点 0 以外的接口或端点，则设备会响应请求错误。

Configured state：这是一个有效的请求。

5.5.10 Set Interface

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None

某些 USB 设备的配置具有互斥设置的接口。此请求允许主机选择所需的备用设置。如果设备仅支持指定接口的默认设置，则可能会在请求的状态阶段返回 STALL。 此请求不能用于更改已配置接口集（必须使用 SetConfiguration（）请求）。

如果接口或备用设置不存在，则设备将响应请求错误。如果 wLength 非零，那么设备的行为没有被指定。

Default state：在设备处于默认状态时收到此请求时的设备行为未指定。

Address state：必须返回请求错误。

Configured state：这是一个有效的请求。

5.6 描述符(Descriptor)

USB 描述符也可以看作是 USB 设备的身份证明。设备描述符的分类见上表 9-6。USB 规范中定义了以下几种描述符：Device(设备)、Device_Qualifier(设备限定)、Configuration(配置)、Other_Speed_Configuration(其他速度配置)、Interface(接口)、Endpoint(端点)、String(字符串)。部分描述符间的关系如下图所示：

5.6.1 设备（Device）

设备描述符描述有关 USB 设备的一般信息。它包含全局适用于设备和所有设备配置的信息。**USB 设备只有一个设备描述符**。

具有全速和高速不同设备信息的高速设备也必须具有 device_qualifier 描述符。所有 USB

设备都有一个默认控制管道。设备描述符中描述了设备默认控制管道的最大数据包大小。配置描述符中描述了特定于配置及其接口的端点。配置及其接口不包含默认控制管道的端点描述符。除了最大数据包大小之外，默认控制管道的特性由该规范定义，并且对于所有 USB 设备都是相同的。默认控制管道的数据包的长度(端点 0 长度)是在设备描述符中定义，而不像其他端点在端点描述符中定义。设备描述符共 14 个字段，长 18 Byte。

Offset	**字段名称**	**长度 Byte**	**字段值**	**意义**
0	bLength	1	0x12	设备描述符的字节数大小,18 字节
1	bDescriptorType	1	0x01	设备描述符类型编号。具体见 4.3.1.2
2	bcdUSB	2	0xJJMN	USB 版本号。格式为：JJ.M.N（JJ-主要版本号，M-次要版本号，N-次次要版本号）定义的 bcdUSB 字段的值为 0xJJMN，例如版本 2.1.3 以值 0x0213 表示，版本 2.0 用值 0x0200 表示。
4	bDeviceClass	1	类	Class 代码（由 USB-IF 分配）。如果该字段重置为零，则配置中的每个接口指定其自己的类信息，并且各个接口独立运行。如果此字段设置为 1 到 FEH 之间的值，则设备支持不同接口上的不同类规范，并且接口可能无法独立运行。 该值标识用于聚合接口的类定义。如果为 FFH 表示厂商自定义
5	bDeviceSubClass	1	子类	Subclass 代码（由 USB-IF 分配）。受 bDeviceClass 字段的值限定。如果 bDeviceClass 为 0，这此值必须为 0。如果 bDeviceClass 字段未设置为 FFH，则所有值都保留给 USB-IF 分配。
6	bDeviceProtocol	2	协议	协议代码（由 USB-IF 分配）。
7	bMaxPacketSize0	1	控制传输端点 0 包大小	端点 0 最大包大小，只能是 8、16、32 或 64。如果设备以高速运行，则必须为 64，表示最多 64 字节的数据包。 高速操作不允许控制端点（端点 0）其

				他最大数据包大小。
8	idVendor	2	**VID**	厂商编号（由 USB-IF 分配）
10	idProduct	2	**PID**	产品编号（由 USB-IF 分配）
12	bcdDevice	2	BCD 码	以二进制编码的十进制表示的设备出厂编号
14	iManufacturer	1	索引	描述厂商字符串索引。当值为 0 时，表示没有厂商字符串，**当为其他值时，主机会利用这个索引值来获取相应的字符串**。
15	iProduct	1	索引	描述产品字符串的索引。当值为 0 时，表示没有产品字符串，**当为其他值时，主机会利用这个索引值来获取相应的字符串**。
16	iSerialNumber	1	索引	描述设备序列号字符串的索引。当值为 0 时，表示没有序列号字符串，**当为其他值时，主机会利用这个索引值来获取相应的字符串。**
17	bNumConfigurations	1	配置描述符个数	bNumConfigurations 字段标识设备支持的配置数量。仅表示当前运行速度下的配置数量。如果特定速度的设备有特定配置，则 bNumConfigurations 字段仅反映单个速度的配置数量

注意：设备描述符中的第 14-16 个字节是三个索引值，分别为厂商字符串索引、产品字符串索引和设备序列号字符串索引。索引值从 1 开始递增。（索引值为 0 表示语言 ID）。在主机获取字符串描述符时，**设备根据主机下发的标准请求的 wValue 中的索引值，返回相应的字符串描述符**。

5.6.2 设备限定（Device_Qualifier）

device_qualifier 描述符描述有关高速设备的信息，如果设备以另一种速度运行，该信息将会改变。例如，如果设备当前正在全速运行，则 device_qualifier 会返回有关如何以高速运行的信息，反之亦然。

Table 9-9. Device_Qualifier Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of descriptor
1	*bDescriptorType*	1	Constant	Device Qualifier Type
2	*bcdUSB*	2	BCD	USB specification version number (e.g., 0200H for V2.00)
4	*bDeviceClass*	1	Class	Class Code
5	*bDeviceSubClass*	1	SubClass	SubClass Code
6	*bDeviceProtocol*	1	Protocol	Protocol Code
7	*bMaxPacketSize0*	1	Number	Maximum packet size for other speed
8	*bNumConfigurations*	1	Number	Number of Other-speed Configurations
9	*bReserved*	1	Zero	Reserved for future use, must be zero

5.6.3 配置（Configuration）

配置描述符描述了关于特定设备配置的信息。描述符包含一个 bConfigurationValue 字段，该字段的值用作 SetConfiguration()请求的参数时，会使设备采用所描述的配置。描述符描述配置提供的接口数量，每个接口可以独立运行。配置完成后，设备可能会对配置进行有限的调整。如果某个特定接口具有备用设置，则可以在配置后选择备用设备。

Offset	Name	Length (Byte)	Value	意义
0	bLength	1	0x09	配置描述符的字节数大小
1	bDescriptorType	1	0x02	配置描述符类型编号 2，见 5.4.1.2
2	wTotalLength	2	数字	以 2 字节二进制数为内容的字段.该字段表示该配置所返回的所有描述符(包括配置、接口和端点描述符) 的大小总和。
4	bNumInterfaces	1	接口描述符个数	此配置所支持的接口数量
5	bConfigurationValue	1	数字	SetConfiguration 命令需要的参数值
6	iConfiguration	1	索引	描述该配置的字符串的索引值
7	bmAttributes	1	位图	配置特征。供电模式的选择，

				D7 保留固定为 1; D6 为 1 表示自供电,值为 0 表示总线供电; D5 为 1 表示支持远程唤醒,值为 0 则不支持; D4～D0 没有意义固定为 0
8	**MaxPower**	1	字段值 *2(mA)	设备从总线提取的最大电流(<=500mA)，**单位为 2mA**。如果需要 200mA，该值为 100.

5.6.4 其他速度配置（Other_Speed_Configuration）

Table9-11 显示的 other_speed_configuration 描述符描述了高速设备的配置，如果它正在以其他可能的速度运行。other_speed_configuration 的结构与配置描述符相同。

Table 9-11. Other_Speed_Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of descriptor
1	*bDescriptorType*	1	Constant	Other_speed_Configuration Type
2	*wTotalLength*	2	Number	Total length of data returned
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this speed configuration
5	*bConfigurationValue*	1	Number	Value to use to select configuration
6	*iConfiguration*	1	Index	Index of string descriptor
7	*bmAttributes*	1	Bitmap	Same as Configuration descriptor
8	*bMaxPower*	1	mA	Same as Configuration descriptor

5.6.5 接口（Interface）

接口描述符描述配置中的特定接口。一个配置提供一个或多个接口，每个接口具有零个或多个端点描述符，用于描述配置中的一组唯一端点。 当配置支持多个接口时，特定接口的端点描述符将遵循 GetConfiguration（）请求返回的数据中的接口描述符。 **接口描述符总是作为配置描述符的一部分返回**。接口描述符不能通过 GetDescriptor（）或 SetDescriptor（）请求直接访问。

Offset	域	Length (Byte)	Value
0	bLength	1	描述符长度，9 字节
1	bDescriptorType	1	描述符类型，该值为 0x04

2	bInterfaceNumber	1	接口编号。一个配置有多个接口时，从 0 开始递增
3	bAlternateSetting	1	接口备用编号，很少使用，设置为 0
4	bNumEndpoints	1	端点数，为 0 即无非 0 端点，只使用默认的控制端点
5	bInterfaceClass	1	
6	bInterfaceSubClass	1	
7	bInterfaceProtocol	1	
8	iInterface	1	该接口的字符串的索引值，若该值为 0，则无字符串

5.6.6 端点（Endpoint）

用于接口的每个端点都有自己的描述符。该描述符包含主机为确定每个端点的带宽需求所需的信息。**一个端点描述符总是作为配置描述符的一部分返回**。端点描述符不能通过 GetDescriptor()或 SetDescriptor()请求直接访问。从不会有端点零的端点描述符。.

Offset	域	Length (Byte)	Value
0	bLength	1	描述符长度，7 字节
1	bDescriptorType	1	描述符类型，该值为 0x05
2	bEndpointAddress	1	端点地址：D0~D3—:端点号；D4~D6—保留；D7-传输方向，1 表示输入，0 表示输出
3	bmAttributes	1	端点属性 **D1~D0**：00-控制传输；01-同步传输；10-批量传输；11-中断传输。 若不是同步传输，则 D2~D5 保留为 0； 若为同步传输： **D3~D2**：00-无同步；01-异步；10-适配；11-同步 **D5~D4**：表示用途，00-数据端点；01-反馈端点；10-含反馈的数据端点；11-保留值 **D7~D6**：保留
4	wMaxPackeSize	2	最大包长度。 全速和低速，D10~D0-最大包长，其他位保留为 0

			高速模式，D12~D11-每个帧附加的传输次数
6	bInterval	1	端点查询的时间。 **全速/高速模式，同步传输**的端点，该值范围为 1-16，该值作为 2 的指数，bInterval 为 4 表示时间段 8； **全速/低速，中断传输**的端点，该值范围为 1-255 **高速，中断传输**，该值作为 2 的指数 **高速，批量/控制，输出**端点，该值范围为 0-255，表示每间隔 bInterval 个微帧，产生一个 NAK

5.6.7 字符串（String）

字符串描述符是可选的。如果设备不支持字符串描述符，则必须将设备，配置和接口描述符中对字符串描述符的所有引用重置为 0。否则在枚举过程中，USB 主机会尝试去获取字符串描述符，如果没有，枚举就会失败。

字符串描述符使用由 UNICODE 编码定义的 Unicode 标准。USB 设备中的字符串可能支持多种语言。请求字符串描述符时，请求者使用由 USB-IF 定义的十六位语言 ID(LANGID)指定所需语言。目前定义的 USB LANGID 列表可以在 http://www.usb.org/developers/docs.html 找到。所有语言的字符串索引 0 返回一个字符串描述符，其中包含设备支持的两个字节的 LANGID 代码数组。USB 设备可能会省略所有字符串描述符。省略所有字符串描述符的 USB 设备不得返回一个 LANGID 代码数组。LANGID 代码数组不是以 NULL 结尾的。数组大小（以字节为单位）是通过从描述符的第一个字节的值中减去 2 来计算的。

语言 ID 描述符结构

Offset	Name	Length (Byte)	Value
0	bLength	1	该描述符长度
1	bDescriptorType	1	描述符类型，该值为 0x03
2	wLANGID[0]	2	语言 ID 号 0
…	…	…	…
2*n+2	wLANGID[n]	2	语言 ID 号 n

字符串描述符结构

Offset	Name	Length (Byte)	Value
0	bLength	1	该描述符长度
1	bDescriptorType	1	描述符类型，该值为 0x03
2	bString	N	UNICODE 编码的字符串

6 USB Hub

略。

7 参考资料

1. usb_20_protocol

2. USB in a Nutshell

3.圈圈教你玩 USB 刘荣 编著