1. 、异步串行通信通常采用的是起止式异步协议，其特点是一个字符一个字符的传输，并且传输一个字符总是以起始位开始，以停止位结束，字符之间没有固定的时间间隔要求。若没有字符传输，则信号的电平处于空闲位，长度不定。由于采用起始位来表示字符传输的开始，并在每一次传输时都要以此来重新核对收发双方的同步。若是接收设备和发送设备两者的时钟频率略有偏差，也不会因偏差的累积而导致错位。但是起止式协议一般用于速率较慢的场合(低于19.2Kb/s)。对于要求更高速率传输时，一般要采用同步协议。
2. 同步协议又有面向字符（Character-Oriented）和面向比特以及面向字节计数三种。以面向字符计数协议为例。其特点是一次传输由若干个字符组成的数据块，而不是只传输一个字符，并规定了10个特殊字符作为这个数据块的开头与结束的标志以及整个传输过程的控制信息，它们叫做通信控制字。协议中设置转义字符，用于指示此时应当把特定的字符看作是数据字符。同步协议虽然传输速率更高，但是控制处理也比异步协议要复杂。
3. 、串行通信的第三个特点是对信息的逻辑定义与TTL不兼容，因此需要进行逻辑电平交换。

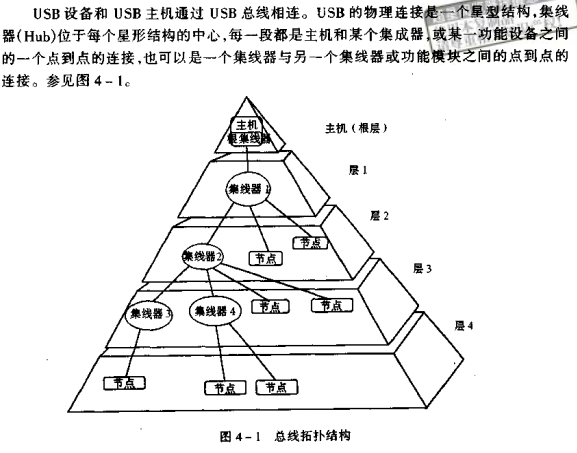
主机负责发现总线上新插入的USB设备，并在拔下某一个USB设备以后释放先前为他们分配的系统资源等操作。在所有的有关于USB技术的概念中，管道和端点是两个基本的概念。在USB主机上的一个软件功能和一个USB设备之间建立一个虚拟连接称为”管道“。在USB技术当中有两种类型的管道----流管道和消息管道。”流管道“是指没有确定的总线帧结构而以数据流的方式进行数据传输的”管道“。”消息管道“中的数据具有一定的帧结构。因而其数据传输就可以与所需的带宽、传输类型和端点特征（传输方向和缓冲区大小）相适应。

管道是在一个USB设备插入系统以后，由位于USB主机上的软件建立的。所有的USB设备都必须要支持”管道0“从而使得USB主机可以利用该管道对USB设备进行配置。

4.4 总线协议

总线上的所有的处理都包括最多三个分组的传输。每一次处理操作开始时，都是由USB主控制器根据一个计划的步骤，发送一个用于描述处理类型和方向、USB设备地址、端点号的USB分组，这一分组被当作令牌分组。被寻址的USB设备通过对恰当的地址域进行解码就可以知道这是发送给自己的分组。对于一个特定的处理操作而言，数据总是由主机传向USB设备或者是由USB设备传向主机。这种数据传输方向在令牌分组中加以规定。然后处理操作的信源就可以发送数据分组或者是指出它自己没有数据需要发送。通常情况下，信宿通过”握手分组“来指明这一次的传输是否成功。

USB总线上这种在USB主机和设备端点之间的数据传输模型称之为”管道“。管道还同数据传输带宽、传送服务类型和传送缓冲区的大小这类的端点特性相联系。只要某一个USB设备完成了配置之后，就会存在管道0.当一个USB设备上电以后，控制管道0这一消息管道就总是存在的，因为这个管道要提供对设备的配置、状态、和控制信息的访问。



4.6.1 插入USB设备

所有的USB设备都可以通过集线器上的特定的端口而接入USB总线。集线器在各个端口的状态中指出USB设备的接入或者是拆除。主机向USB集线器进行询问来决定其指示的原因。集线器会作出响应，指明已经用于连接USB设备的端口。主机通过控制管道，使用缺省的USB地址来使这些端口可用，并对USB设备进行寻址。所有的USB设备在刚刚接入USB总线或者是复位以后，都要利用USB缺省地址来寻址。主机利用已分配的USB地址和端点号”0“来建立该USB设备的控制管道。

4.6.2 拆除USB设备

当集线器的一个端口移除某一个USB设备的时候，集线器会自动的废除该端口并向主机发出设备移除的指示。而主机会从主机的数据结构当中删除该USB设备的信息。

如果移除的是一个集线器，则会先对该集线器接入的所有USB设备执行移除操作过程。

4.6.3 总线枚举

总线枚举是指对总线上接入的USB设备进行识别和寻址操作。由于USB技术允许在任意的时刻从USB总线上插入或者是拔出USB设备，因此，USB总线的总线枚举操作是一种持续执行的工作。而且，USB总线的枚举还包括发现和处理设备的移除。

4.6.4 层间关系

USB设备在逻辑上可以分成一个USB设备接口部分，一个设备部分和一个功能部分。而USB主机在逻辑上可以分成USB主机接口部分，集合系统软件部分（USB系统软件和主系统软件）和设备软件部分。

4.7 数据流类型

USB协议支持以单向或者是双向的方式，在USB主机和一个USB设备之间交换功能数据和控制信息。USB的数据传输是在主机软件和一个USB设备上的特定端点之间进行的。一个给定的USB设备支持多个数据传输端点。USB主机将会分别的处理一个USB设备的任意端点和其他端点上的通信。这种主机软件和一个USB设备端点之间的联系称为管道。比如，一个给定的USB设备可以有一个端点用来支持一个接收数据的管道。同时由另外的一个端点支持一个发送数据的管道。

在USB的体系结构中有四种基本的数据传送类型：

1. 控制信息传输：用于在设备接入时对其进行配置，也可以用于其他目的
2. 批量数据传输：在数据相对较多和突发数据量较大的时候使用
3. 中断数据传输：主要用于向主机通知设备的服务请求，它是由设备发起的通信，具有数据量小，非周期，低频率，延时一定等特点。
4. 同步或流实时数据传输：占用预先商量好的带宽，并且有预先商量好的发送延时。

4.7.5 分配USB带宽

USB带宽要在管道中进行分配，当一个管道建立以后，USB可以为一些管道分配带宽。而USB设备则要为数据提供一些缓存能力。这里假定需要贷款越大的USB设备可以提供更大的缓冲区。USB体系结构的设计目标就是需要缓冲操作来给予减小的硬件时延限制在几个毫秒之内。

4.8 USB设备

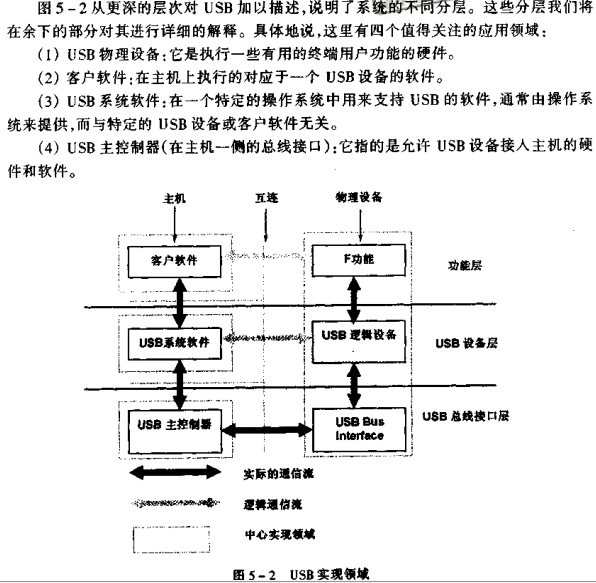
4.8.1 设备特征

所有的USB设备都要通过一个唯一的USB地址对其进行访问。每一个USB设备还要支持一个或者是多个端点，用以实现和主机之间的通信。而且所有的USB设备都需要支持一个特殊的端点0，该端点是USB控制管道在USB设备一侧的接入点。

与端点0相对应的是用来完整描述该USB设备的信息。这些信息包括以下的几类：

1. 标准。这是一类对所有的USB设备都通用的定义，并且包含厂商的标识，设备类型和电源管理等项目，对设备、配置、接口、端点的描述携带了与该设备的配置有关的信息。
2. 类型。该信息的定义将根据USB设备的类型而不同
3. USB设备供应商。USB设备供应商可以自由的提供此处所需要的信息。但是，本规范并未对其格式作出规定。

而且，每一个USB设备都携带了USB控制和状态信息。所有的USB设备都支持通过其USB控制管道的通用访问模式。



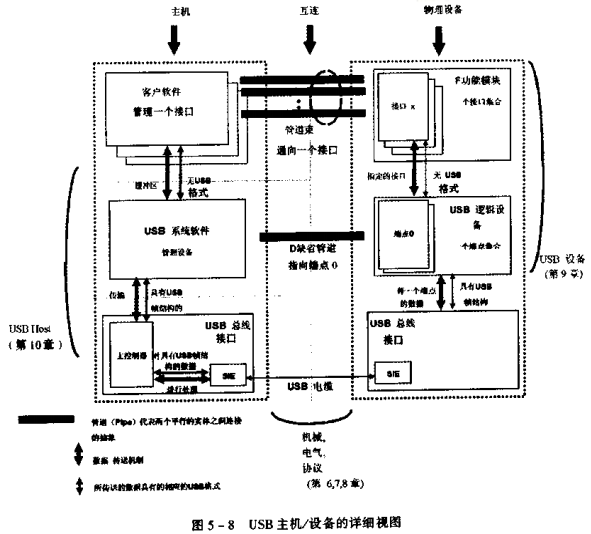
为了协 助主机识别和配置USB设备，每一个设备都要携带和报告与配置相关的信息。对于所有的逻辑设备而言，所报告的信息中有一些是通用的。而另外的一些则是由设备的功能所决定的

5.2.3 物理总线拓扑结构

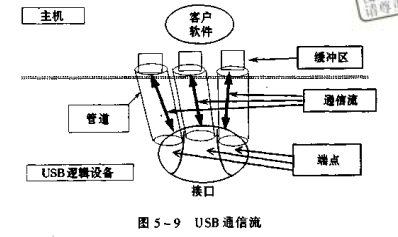
主机中包含一个嵌入的集线器，它称之为根集线器。通过根集线器，主机可以提供一个或者是多个接入点。

5.3 USB通信流

USB为主机上的软件和客户的USB功能模块之间提供通信服务，不同的功能模块会对通信流有不同的要求，需要你不同的客户来实现相互作用。通过允许将不同的USB功能模块的不同通信流分离开来，USB提供了更好的整体总线利用率。每一个通信流都要使用某一个总线访问来完成客户和功能模块之间的通信，并且终止于设备上的某一个端点。设备的端点用于区别任意的通信流。



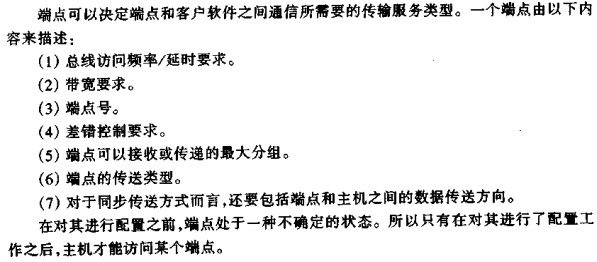
一个逻辑设备对于整个系统而言就是一个端点的集合。根据其使用的接口，端点可以被分成不同的端点集。接口是对功能的认识。系统软件使用缺省的管道（与端点0对应）来管理USB设备。而客户软件使用的管道束（与一个端点集对应）。客户软件要求数据在主机上的缓冲区和USB设备上的端点之间，通过USB总线来移动。主控制器（或者是依赖于传输方向的USB设备）对数据进行分组，然后通过USB来传输。当使用总线访问来实现USB上的数据传输的时候，主控制器也会协同工作。



主机上的软件通过一系列的通信流来与一个逻辑设备进行通信。这一系列的通信流由设备软件/硬件设计者来加以选择，从而使得USB设备的通信要求与USB提供的传输特性更有效的匹配。

5.3.1 设备端点

端点是一个USB设备唯一可以确认的部分，它是主机和设备之间的通信流终点。每一个USB逻辑设备都包含了一个相当独立地进行操作的端点集合。软件只能通过一个或者多个端点与一个USB设备通信。在设备接入的时候，每一个逻辑设备都有一个由系统分配的唯一的地址。而一个设备上的任意一个端点都有一个由设备而定的（设计时）唯一标志和端点号，利用设备的地址和端点号就可以唯一的指定一个端点。



1. 端点0要求
   1. 该端点用于对一个逻辑设备进行初始化和一般的操作（即对一个逻辑设备进行配置）。端点0提供了对设备配置信息的访问权，通过它还允许访问一般的USB状态和控制操作
2. 非0端点要求
   1. 除端点0以外，低速设备有两个端点可以供选择。对于全速设备，它附加的端点数仅仅受到协议的限制，最多有16个输入和16个输出端点。

一个端点只有在对其进行配置以后才能够使用，包括端点0在内的所有端点，都作为设备配置过程中的普通部分来对其进行配置。

5.3.2 管道

一个USB管道是设备上的一个端点和主机上的软件的联合体。管道表示经过一个存储器缓冲区和一个设备上的端点，可以在主机上的软件之间传输数据的能力。

管道需要确定以下的三个参数：

1. 对USB总线访问和宽带使用的声明。
2. 传输类型
3. 所对应的端点的特征，如方向和最大的数据负载尺寸等，数据负载尺寸是指总线操作中，一个数据分组的数据域中所携带的分组。

对一个USB设备进行配置以后，就会形成一个管道。由于一个USB设备上电以后总是要对端点0进行配置，所以端点0总是拥有一个管道。称为缺省管道。系统软件利用该管道来确定设备的识别和配置要求，并对设备进行配置。当设置完毕以后，该设备的专用软件也可以使用缺省管道。但是USB系统软件却也保留对缺省管道的所有权，并且由它来协调另外的客户软件对该管道的使用。

一个客户软件通常是通过向一个管道发出IRP来要求传输数据，发出IRP之后客户软件就等待或者得到指示。当与一个IRP相对应的总线操作完成以后，无论其成功与否，这个客户软件都会被告知IRP已经处理完成了。

如果一个管道没有未处理或者是正在进行的IRP，那么该管道就处于空闲状态，主控制器不会对该管道进行任何的操作，即这一管道的端点没有任何的总线处理指向它。只有当这个管道中有没有处理的IRP时，该管道中才会有总线操作。

为了通过总线来传输数据，一个IRP可能需要多个数据负载。对于一个需要多个数据负载的IRP来说。其分组的数据域应该具有最大的尺寸，直到最后一个数据载荷中包含该IRP中余下的部分。对于拥有输入的短分组（即数据小于数据载荷的最大值）的IRP而言，其数据不能够填满该IRP的缓冲区，我们可以根据用户的要求而采用两种方法处理。

由于主控制器必须要在两种不同的情况下进行不同的操作，而且它不会知道对于一个给定的IRP应当要采用哪一种的处理方式，所以必须要告诉每一个IRP用户所要求的操作方式。

USB系统软件确保多个请求不会同时送到一个端点。在某一个时刻一个端点仅仅需要服务于单个消息请求。主机上的多个客户软件可以通过缺省管道发出请求，但是这些请求却是按照先进先出的顺序送至某一个端点的。

5.4 传输类型

USB通过位于同主机上的客户软件相对应的内存缓冲区和USB设备上的端点之间的管道来传输数据。USB规定对于任意一种管道类型（流或消息）数据经过总线时都要进行分组，但是最终对一个总线操作的数据负载中传递的数据进行分析和解释的工作，则是要由客户软件和使用该管道的功能模块来完成的。

5.5 控制传输

控制传输允许对一个设备的不同部分进行访问，它用于支持客户软件及其功能模块之间的配置/命令/状态等类型的通信流。一个控制传输由以下的几个部分构成：总线建立操作，负责将请求信息由主机送至功能模块；零个或者是多个数据处理操作，传输方向由建立操作指定。状态处理操作，用于功能模块向主机返回状态信息。

控制传输只能由消息管道来实现。因此，使用控制传输的数据流要遵循特定的数据结构。

5.8 批量传输

批量传输类型是设计用来支持那些在不同的时刻需要传输相对而言数据量较大的数据，并且该传输可以推迟到有可用带宽再进行的设备。申请一个批量传输类型的管道需要提供以下条件：

1. 根据可用的带宽对USB设备进行访问
2. 如果由于总线上的错误而出现偶然的传输失败，需要重试。
3. 保证对数据的传输，但是不对带宽或时延提供保证。

批量传输仅仅会在有可用的带宽的基础上进行，对于一个具有大量的空闲带宽的USB系统来说，批量传输可能会更快一些；如果一个USB系统只有很少的可用带宽，那么批量传输就会缓慢的进行。

批量管道是一个流管道，管道中的 通信流不具有USB规定的数据结构，一个给定的管道当中总是有通信流进入或流出主机。如果一个设备要求双向的批量通信流，就必须要使用两个批量管道，一个方向一个。

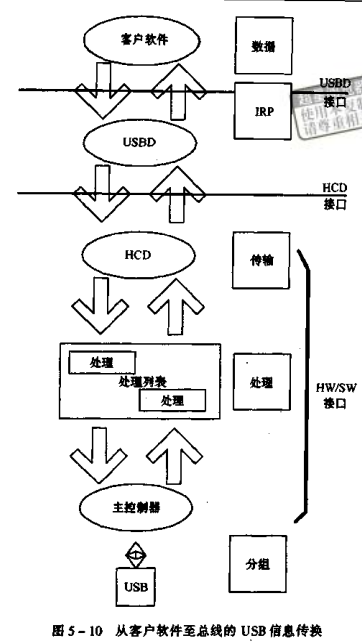
用于批量传输的一个端点要说明它可以从总线上接收或是向总线发送的最大数据负载的大小。在配置的过程中，USB系统软件将读取端点的最大数据负载值并且保证不会向该端点发送超过其所能支持的最大数据负载尺寸的数据负载。

只有全速率设备可以使用批量传输，一个端点不能指出一个批量管道所要求的总线访问频率，USB将会负责平衡所有的批量管道和某一些未处理的IRP的总线访问请求。

5.9.1 传输管理

为了让处理操作通过总线，传输管理包括几个对不同对象进行操作的实体：

1. 客户软件：通过调用/回调（callback）来向USBD接口请求IRP，使用/产生某一个来自/送往一个功能端点的功能数据
2. USB驱动程序（USBD）：通过对适当的HCD的调用/回调，对客户IRP中来自或送往设备端点的数据进行交换。
3. 主控制器驱动程序（HCD）：对送往或是来自处理操作的IRP进行转换（根据主控制器的应用要求）并且对这些IRP加以组织以便用于主控制器的操作。主控制器驱动程序和它的硬件之间的交互依赖于实际的应用。



1.客户软件：决定需要为一个功能模块建立哪一种类型的传输。它使用某个适当的操作系统接口来请求IRP。客户软件只清楚操作它的功能模块所需要的管道集（即接口）。客户软件的请求需要通过USBD接口来提出。所有提交的IRP必须要遵守先前协商好的对带宽的限制，这一限制在设备接入总线和对其进行配置操作时已经设定好了。如果一个功能模块从一个非USB环境进入到一个USB环境，驱动程序将利用存储器或者是I/O访问功能来直接操作功能硬件，它是为了操作其功能模块而与USBD相互联系的最底层的客户软件。在客户软件为其功能模块请求了一个传输并且该请求得到服务之后，客户软件会得到IRP已经完成的通知。如果传输中包含了向主机传输数据的功能，客户软件就会访问与某个完成了的IRP相对应的数据缓冲区的数据。

2. USB驱动程序：在一个接入总线的配置操作和正常的传输中，USBD通常要对总线访问进行两次斡旋。当一个设备插入并对其进行配置时需要USBD来保证总线上可以容纳所要求的设备配置操作。USBD从用来说明所需要的设备配置操作的配置软件中接收配置请求，其内容包括端点、传输类型、传型周期、数据大小等。根据可用的带宽和总线上对请求类型的容纳能力，USBD将接受或拒接一个配置请求。若接受，USBD将会为发出请求的设备创建一个所需类型的管道，该管道具有为这种传输类型所定义的适当的限制。一旦对一个设备进行了配置，客户软件就可以为在它和它的功能端点之间传输数据而请求IRP。

3.主控制器驱动程序：HCD负责在操作的过程跟踪IRP，并且保证不超过USB带宽和帧时间的最大值。当为一个管道建立了IRP时，HCD会将他们添加到处理操作列表当中。如果一个IRP完成了，HCD会通知发出该请求的客户软件，该IRP已经完成。如果这个IRP包含从功能模块到客户软件的数据传输，那么数据将会被放置在客户软件指定的数据缓冲区。IRP的定义依赖于所使用的操作系统。，

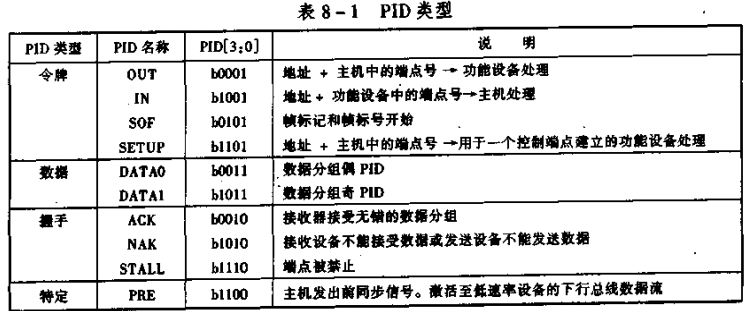
4.处理操作列表：是一个根据主控制器的应用而作出的说明，它描述了那些需要在总线上运行的，当前仍然没有处理完成的总线处理操作集。根据某个与主控制器的应用有关的表示方式，一个典型的处理操作列表包含了一系列的帧说明。只有HCD及其主控制器可以访问某一个表示方式。每一个帧说明中都包含了处理操作说明，其中指明了像数据大小（字节）、设备地址和端点，以及用来发送或者是接收数据的存储区域等参数。

5.主控制器：访问处理操作列表，并将其转化为总线操作。而且主控制器提供了一种报告机制，由此可以活得一个处理操作的状态（完成、等待处理、禁止等）。主控制器可以将处理操作转变为与应用有关的适当操作，从而使得USB分组可以通过基于根集线器的总线拓扑结构进行传输。

5.9.4 计算功能模块/软件中的缓冲区大小

客户软件和功能模块都需要为那些在总线上等待的未处理的数据处理操作提供缓冲区。对于非同步管道而言，缓冲区空间只需要足够容纳下一个数据分组。如果对于某个端点有多个处理操作请求等待处理，就必须为每一个处理操作提供缓冲处理。一个功能模块可能需要能够用来绝对精确的计算最小缓存时间的办法，这是因为USB规范并未对功能模块及客户软件的交互进行规定。

1. 协议层

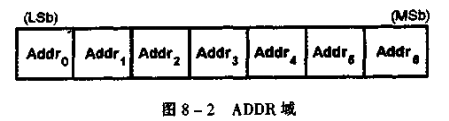


8.3.2 地址域

可以利用两个域对功能设备的端点进行寻址：功能设备地址域和端点域。一个功能设备需要能够对地址和端点域进行完全的解码。不允许将地址或端点混淆，任一域内出现的不匹配都会使得这一令牌被忽略。同样，访问一个未被初始化的窗口也会使得令牌被忽略。

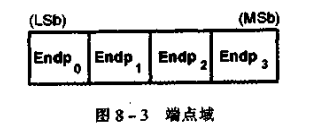
功能设备的地址（ADDR）域通过其地址对功能设备加以区别，它可以利用令牌PID的值来说明该功能设备是一个数据分组的发送者还是接收者。ADDR域可以指定用于IN、SETUP和OUT令牌。根据规定，每一个ADDR值都定义了一个功能设备。当复位和上电时，设备的地址缺省为0.并且必须在枚举的过程中由主机来对其进行编程。缺省的地址0为缺省状态所保留，不能够将其分配给一般的操作使用。

如下图所示ADDR<6:0>可以说明全部的128个地址



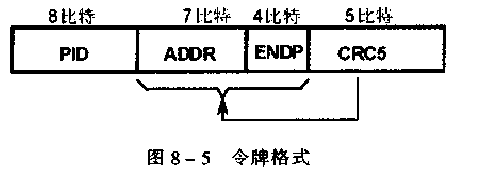
8.3.3 端点域

一个附加的4比特端点（ENDP）域允许那些需要不止一个子信道的功能模块进行更灵活的寻址。端点号视功能设备而定。规定端点域只能供IN、SETUP、OUT令牌的PID使用。所有的功能设备必须支持一个控制端点0.低速率设备中的每一个功能模块最多可以支持两个端点地址：0和另外一个端点。全速率功能模块最多可以支持16个端点。



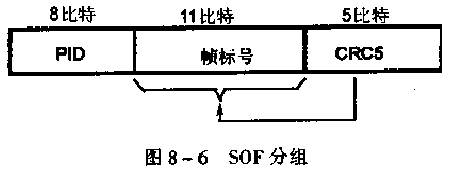
8.4.1 令牌分组

下图给出了一个令牌分组的域格式．一个令牌包括一个用于区别是ＩＮ／ＯＵＴ／ＳＥＴＵＰ分组类型的PID域,ADDR以及endp域.对于OUT和SETUP处理操作而言,地址和端点域可以唯一的标识一个即将接收后继数据分组的端点.对于IN处理操作而言,这些数据域将唯一的标志一个即将发送一个数据分组的端点.只有主机才能发送令牌分组.INPUT定义了一个由一个功能模块至主机的数据处理操作.OUT和SETUP则定义由主机至一个功能模块的处理操作



8.4.2 帧开始分组(SOF)

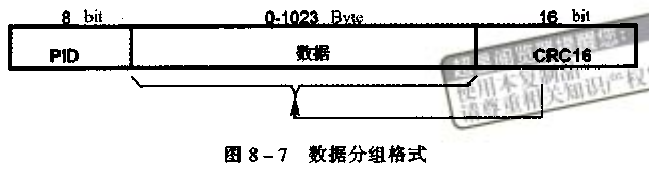
SOF分组包含一个用于指示分组类型的PID,后面还跟有一个11比特的帧标号域.



SOF令牌由仅对令牌进行处理的操作组成,它在依据每一帧的开始而精确定时的时间间隔内分配一个帧开始标记和帧标号.SOF令牌不会使任何进行接收的功能设备产生一个返回分组,因此,不能够保证SOF可以传送到某一个给定的功能模块.

8.4.3 数据分组

如下图所示:一个数据分组包括一个PID,一个数据域和一个CRC域.不同的PID可以区分两种类型的数据分组:DATA0和DATA1.



数据必须以整个字节进行发送,数据CRC仅对分组中的数据域进行计算,而不包括PID,因为PID有自己的校验域.

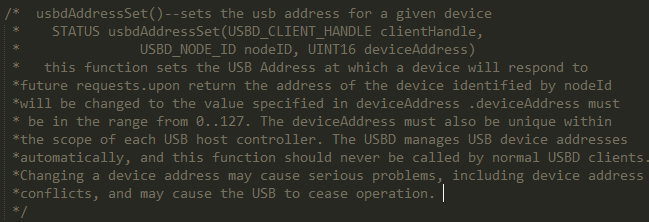
8.5.1 批量处理操作

批量处理操作类型的特性是:通过差错监测和重试方法,它可以保证在主机和一个功能设备之间无差错的数据传输.批量处理操作利用了一个具有三个阶段的处理操作,包括令牌/数据/握手分组,

1. USB设备结构

9.1.1 可见的设备状态

1. 连接:一个USB设备可以接入USB,也可以从USB上拆除.当一个USB设备从USB上拆除后所处的状态.规范并未定义。
2. 缺省:在设备上电以后,直到它从总线上接收到了一个复位信号以后,它才能对任一总线处理操作做出响应.在设备收到了一个复位信号以后,设备可以使用缺省地址来对其进行寻址.
3. 地址分配:当设备刚上电或是复位以后,所有的设备都是用缺省地址.主机在连接或者是复位操作之后,会为每一个USB设备分配一个唯一的地址.当一个USB设备被挂起的时候,它会保留为其分配的地址.无论设备目前已被分配了一个唯一的地址,还是使用的缺省地址,它都能在其缺省管道对请求做出响应.



1. 配置完成:在USB设备可以使用之前,必须要对其进行配置.在设备看来,配置就是向其配置寄存器内写入一个非零的值.对一个设备的配置或是对一个可替代的设置的改变,会使与受影响的接口内的端点有关的所有状态和配置值都设置为其缺省值.这包括利用数据触发器,将任意端点的数据触发值设置为DATA0.
2. 挂起:当设备在一个特定的周期内没有观测到任何的总线操作,他就会自动进入挂起状态.当被挂起时,USB设备保留了包括其地址和配置信息在内的任一个内部状态.

9.1.2 总线枚举

当一个USB设备接入USB或是从USB上拆除时,主机使用一个称之为总线枚举的过程来识别和管理必要的设备状态的变化.当一个USB设备接入以后,将会采取下列的动作:

1. 先在该USB设备所接入的集线器,通过一个其状态变化管道上的回应,可以向主机报告该事件.此时USB设备处于连接状态,而连接他的端口则被禁用.
2. 主机通过询问集线器来确定变化的真实性质.
3. 既然主机已经知道了新的设备所接入的端口,它就会向该端口发送一个端口激活和复位命令
4. USB设备接受端口激活和复位命令,在USB设备就处于上电的状态,它的所有的寄存器和状态就被重新设置了,而且它可以对缺省地址做出响应.
5. 在为该USB设备分配一个唯一的地址之前,利用缺省的地址仍然可以访问其缺省的管道.主机通过读取该设备的描述符,来确定这一个USB设备的缺省的管道实际可以使用的最大数据负载尺寸.
6. 主机向USB设备分配一个唯一的地址,使设备进入寻址状态.
7. 主机通过从0至N读取每一个配置,它可以从设备中读出配置信息.
8. 基于配置信息和如何使用这一个USB设备,主机可以向设备分配一个配置值.这时设备就处于配置完成的状态,并且在这一个配置中的所有端点都具有其描述的特征.

9.2 通用USB设备操作

9.2.1 动态连接和拆除

可以在任意的时刻连接或拆除USB设备.一旦检测到了一个连接操作,主机会激活连接了这一设备的集线器端口.而且会使设备复位.一个复位后的USB设备具有如下的特征:

1. 对缺省的USB地址做出响应
2. 未被初始化
3. 不会首先被挂起

9.2.2 地址分配

当一个USB设备连接以后,主机会在设备被主机复位并且设备的所有集线器端口被激

活以后,负责向该设备分配一个唯一的地址.

9.2.3 配置

一个USB设备必须要在完成配置之后才能够被使用.主机负责对一个USB设备进行配置.主机总是从USB设备获得配置信息,从而确定设备的功能.

在单个配置操作内,一个设备可以支持多个接口.一个接口就是一个相应的端点集.它代表该设备向主机提供的一个单一的特性或功能.用于同这一相应的端点集通信的协议和接口中的每一个短点的用途,都可以作为一个设备类型或者是某个厂商定义的一部分来加以说明.

而且,位于一个配置内的一个接口可以有可供替换的设置,可以对序号或相应的端点的特征进行重新定义.如果是这种情况,设备就会支持或得(Get)接口和设置(Set)接口请求,并为某一个接口报告或者是选择某个可选的设置.

在每一个配置内,每一个接口描述符都包含了可以对该接口号和可替换的设置进行识别的域,接口号可以从零开始,直到比当前该配置所支持的接口数少一个.可替换的设置的计数范围是,从0到比某个接口的可替换设置数少一.当一个设备最初被配置的时候,其缺省的设置是可替换设置0

为了支持可以管理一个相应的USB设备组的自适应设备驱动程序,设备和接口描述符包括了类型,子类和协议域.这些域用来标志一个USB设备所提供的功能和同设备上的该功能模块通信所使用的协议.为一个相应的设备类型分配的类型代码已经作为USB规范的一部分而被定义.一个设备类型可以进一步的划分为子类,而且在一个类型或是子类中,一个协议代码可以规定软件是如何同设备通信的.

9.2.4 数据传输

数据可以以四种方式在一个USB设备端点和主机之间传输.某些端点可能适合不同的数据传输类型.但是,一旦配置完成,一个USB设备端点只能够使用一种数据传输模式.

9.3 USB设备请求

所有的USB设备都会对设备缺省管道上来自于主机的请求做出响应,这些请求是利用控制传输产生的.请求和请求参数则是在建立分组中传送到设备的.

9.5 描述符

USB设备以描述符来报告其属性。一个描述符是具有一个确定格式的一个数据结构。每一个描述符都是以一个一字节宽并且包含了该描述符中的所有字节数的域为开始，在跟上一个说明描述符类型的一字节宽的域。

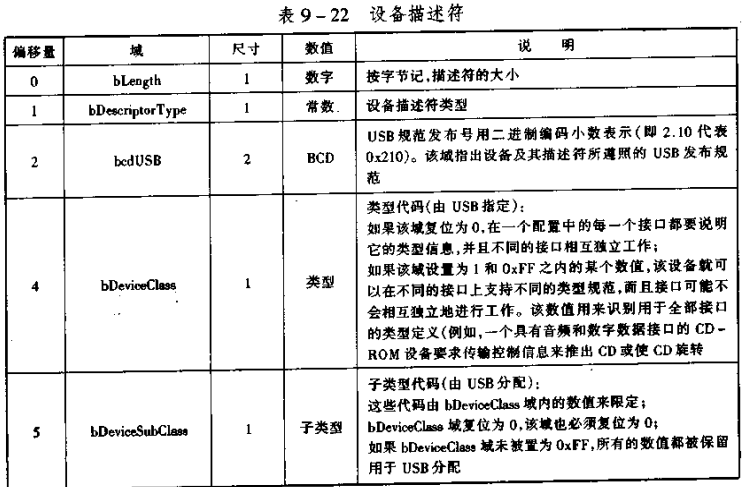
使用描述符可以对单独的配置的属性进行简洁的存储，这是因为每一个配置都可以重用来自于那些具有相同的特征的其它配置的描述符或部分描述符。在这种方式下，描述符与一个关系数据库中的单独的数据记录相类似。

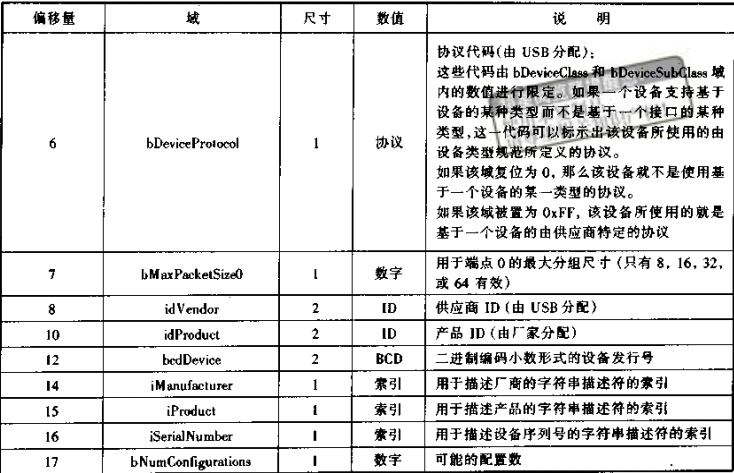
如果一个描述符长度域内的返回值小于规范规定的值，那么该描述符就是不正确的，并且应该由主机来丢弃。若一个描述符长度域内的返回值大于本规范定义的值，主机就会忽略多余的字节，但是下一个描述符却是利用返回的长度而不是所期望的擦汗你孤独来进行定位的。

类型和供应商专用描述符可以以两种方式之一返回。与标准描述符有关的类型和供应商专用描述符，是在一个同标准的描述符相同的数据缓冲区之内，紧跟着相应的标准描述符而返回的。

例如：如果某一个类型或者是供应商描述符同一个接口描述符有关，有关的类型或者是供应商专用的描述符就会放在缓冲区内的接口描述符和接口端点描述符之间。作为对于一个GET\_CONFIGURATION\_DESCRIPTOR请求的响应而返回。一个标准的描述符的长度不会增加用来容纳设备类型或者是某个供应商描述符的扩展。类型或者是供应商专用描述符遵循同样标准描述符一样的格式。与一个标准描述符无关的类型或者是供应商专用描述符，可以利用类型供应商特定的请求而返回。

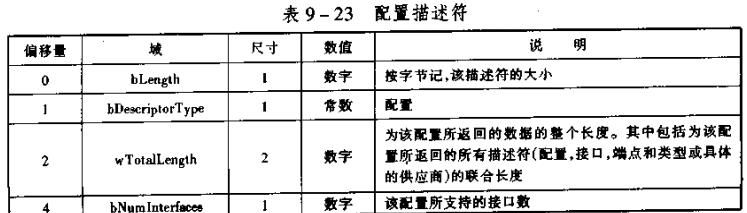
一个设备描述符描述了有关一个USB设备的通用信息。它包括了可以同时应用于设备和所有的设备配置的信息。一个USB设备只有一个设备描述符。结构如下：

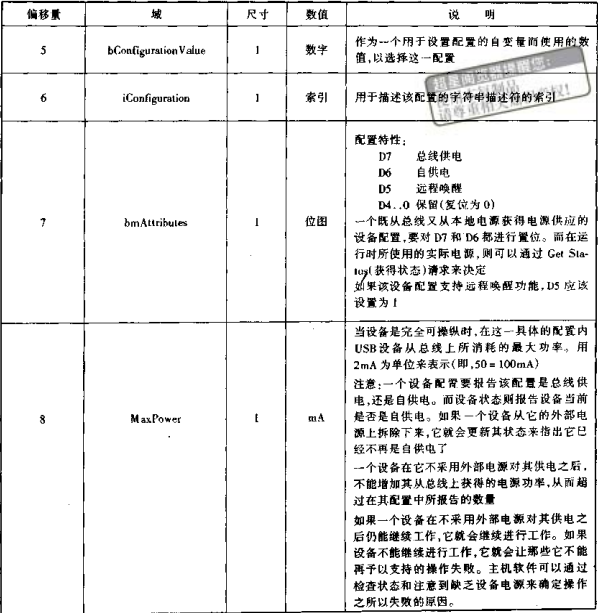




一个设备描述符描述了一个有关USB设备的通用信息。包括可同时应用于设备和所有的设备配置的信息。一个设备缺省管道所使用的端点0所欲允许的最大分组尺寸将在设备描述符中说明。对应于一个配置及其接口的端点将在配置描述符内说明。一个配置及其接口不包括一个用于端点0的端点描述符。

配置描述符描述了有关某个设备配置的信息。这个描述符包括了一个具有一定数值的bConfigureValue域，当其作为一个提供给Set Configuration请求的参数使用时，它可以使设备采用所描述的配置方法。结构如下：



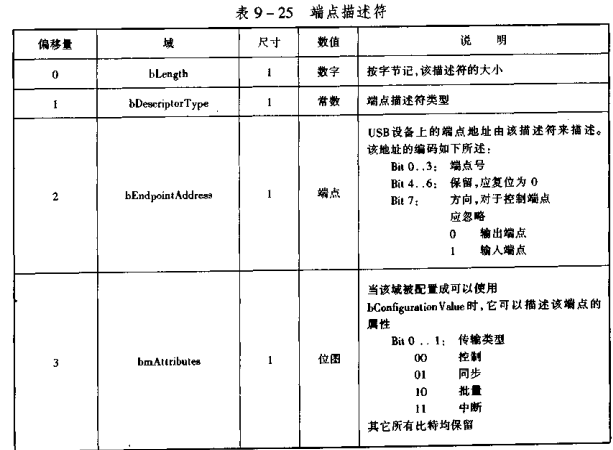


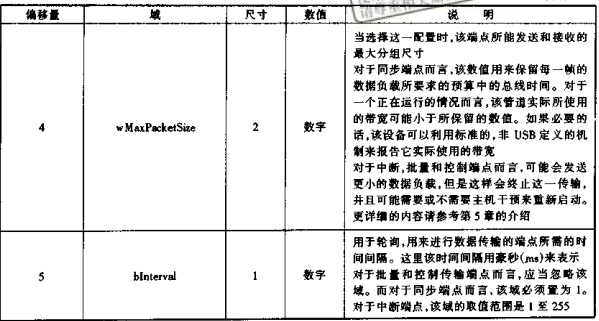
一个USB设备具有一个或者多个配置描述符。每一个配置都有一个或者多个接口，并且每一个接口都有一个以上的端点。在单个配置中，一个端点不会在接口之间共享，除非该端点被同一个接口的可替换的设置所使用。端点可以在那些没有这一限制的具有不同的配置的一部分的接口之间共享。一旦配置完成，设备还可以支持对配置进行有限的调整。如果某个接口有可替换的设置，在配置之后就会选择这个可替换的设置。在一个接口中，一个同步端点所允许的最大分组尺寸也可以被调整。

接口描述符描述了由相应的配置所提供的某个接口。一个配置提供了一个以上的接口，每一个接口都有自己的端点描述符来描述该配置内的一个唯一的端点集。当一个配置支持的接口多余一个时，在Get\_Configuration请求所返回的数据中，某个接口的端点描述符紧跟在接口描述符之后。一个接口描述符总是作为一个配置描述符的一部分而返回的。它不能用一个Get\_Descriptor或者是Set\_Descriptor请求来直接访问。结构如下：

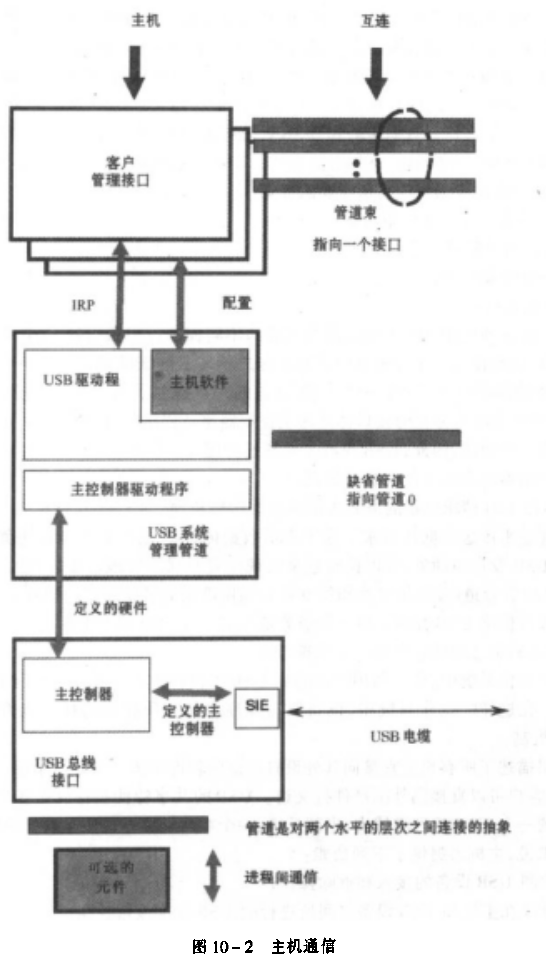


用于一个接口的每一个端点都有自己的描述符。该描述符包括了主机确定每一个端点的带宽请求所需要的信息。一个端点描述符总是作为一个配置描述符的一部分而返回。不能利用Get\_Descriptor或Set\_Descriptor请求来直接对其进行访问。对于端点0而言，它永远不会具有一个端点描述符。





1. USB主机



所有的集线器都提供了一个状态管道,在该管道上可以报告集线器和其端口的变化.这包括对一个设备何时接入其端口,或者是从其端口上拆除的指示.一个USBD客户,通常为集线器驱动程序,将作为集线器状态变化管道的所有者来接收这些指示.对于设备的接入,集线器驱动程序会初始化该设备的配置过程.在某一些系统当中,集线器驱动程序是操作系统为管理设备而提供的主机软件的一部分.

每一个设备都实现了一个或者是多个接口,一个客户利用这些接口来同设备通信,每一个接口都包括0个或者是更多的管道,这些管道可以在客户和设备上的某一个端点之间分别传输数据.USBD会在主机软件的明确要求之下创建接口和管道.当发出控制请求时,主控制器根据主机软件所提供的参数来提供服务.

依据要传输的数据的传输要求,一个管道有几种特性.包括:数据所要传输的速率,是以稳定的速率或者是变化的速率来提供数据,在传送之前数据可以被延迟多久和所传输的数据丢失是否会是灾难性的.一个USB设备的端点需要说明某一个管道所要求的特性,因而端点被作为一个USB设备特征信息的一部分来加以说明.

10.3.1 设备的配置

USB系统采用一个称为集线器驱动程序的一个USBD的专门用户,由它作为一个用于向某一个集线器添加设备和从某一个集线器拆除设备的处理中心.一旦集线器驱动程序接到了这样的指示,它就会以某一个操作系统所特定的方式,利用另外的主机软件和其他的USBD客户来识别和配置设备

当一个设备接入时,集线器驱动程序将接收到检测这一变化的集线器所发来的一个指示信息,集线器驱动程序利用集线器所发来的信息,来请求一个来自于USBD的设备标志.接着USBD为该设备建立一个缺省管道并向集线器驱动程序返回一个设备标识.

现在设备已经为配置做好了准备,对于一个设备而言,必须要在三种类型的配置操作完成了以后,它才能为使用做好准备.

1. 设备配置:包括建立设备的所有USB参数和使用所有该设备可见的主机资源.这些工作是通过在设备上配置配置值而完成的.不需要对设备全部重新进行配置,也可以允许像使用交替的设置这样的有限的配置变化存在.一旦一个设备被配置以后,从他的观点来看他已经为使用做好了准备.
2. USB配置:为了实际建立一个准备由一个客户来使用的USBD管道,客户还必须说明另外的USB信息,而这些信息对于设备而言是不可见的.这一信息称为管道原则,描述客户会如何来使用管道.它包括了客户利用一个IRP所能够传输的最大数据量,客户可以使用的最大服务时间间隔和客户的指示标识等项.
3. 功能配置:一旦配置完类型1和2,从USB的观点来看管道已经完全为使用做好了准备,但是在客户可以实际使用管道之前,可能还要而外的由供应商或设备类型而定义的建立操作.这一配置实在设备和客户之间所进行的独立的操作,并为由USBD对其进行标准化.

实际的设备配置由负责配置的软件来完成,根据操作系统的实现,负责配置的软件包括:

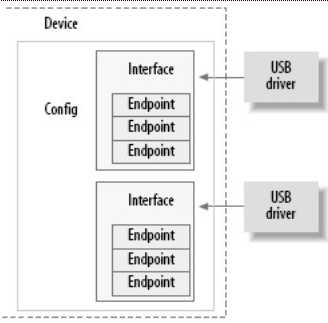
1. 集线器驱动程序
2. 其他主机软件
3. 设备驱动程序

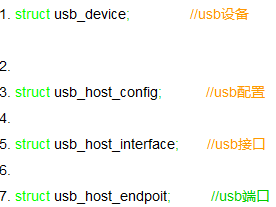
配置软件首先会读取设备描述符,然后会要求对每一个可能的配置加以说明.他可能使用所提供的信息来装载一个客户。

//=============================杂记==================================//

//=============================2017.05.18 BEGIN===========================//

USB驱动的框架主要包括设备、配置、接口和端点几个组成部分，相应的数据结构为：





上图描述的关系如下：

1. .一个USB设备可以有多个配置，这些配置之间可以进行相互切换，某一个时刻只能对应一个配置；
2. 一个USB配置可以有多个接口，一个接口代表了一个基本功能，对应一个USB客户端驱动程序；
3. 一个USB接口可以有多个USB端点。

对于USB规范中定义的需要默认控制端点(endpoint 0)响应的各种请求(如set\_configuration,set\_interface等)，则是通过USB控制器后端的系统软件解析这些请求，并将合适的数据写入FIFO，由USB控制器发送到设备来完成这一请求。所以USB规范中定义的各种描述符(device、configuration、interface、endpoint)是由系统软件来维护的，而不是硬件自动响应的，USB仅仅是一个接口，它用于数据的传输，并不对数据本身进行解析和处理。

//=============================2017.05.18 END=============================//