USB 的重要关键字：

1. 端点：位于 USB 设备或主机上的一个数据缓冲区，用来存放和发送 USB 的各种数据，每一个端点都有唯一确定的地址，有不同的传输特性（如输入端点、输出端点、配置端点、批量传输端点）。
2. 帧：时间概念，在 USB 中，一帧就是 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，每一份就是一个 USB 传输动作。
3. 上行、下行：设备到主机为上行，主机到设备为下行。

Q1：USB 的传输线结构是怎样的？

一条 USB 传输线分别由地线、电源线、D+和 D-构成。D+和 D-是差分输入线，使用 3.3V 电压，与 CMOS 的 5V 电平不同。电源线和地线可为设备提供 5V 电压，最大电流为 500mA，可在编程中设置。

Q2：数据是如何在 USB 传输线里传送的？

数据在 USB 传输线里是由低位到高位发送的。

Q3：USB 的编码方案是怎样的（了解即可）？

USB 采用不归零取反来传输数据，当传输线上的差分数据输入 0 时就取反，输入 1 时就保持原值，为了确保信号发送的准确性，当在 USB 总线上发送一个包时，传输设备就要进行位插入操作（即在数据流中每连续 6 个 1 后就插入一个 0），从而强迫 NRZI 码发生变化。

Q4：USB 的数据格式是怎样的？

USB 数据格式是由二进制数字串构成的，首先数字串构成域（有七种），域再构成包，包再构成事务（IN、OUT、SETUP），事务最后构成传输（中断传输、并行传输、批量传输和控制传输）。下面简单介绍一下域、包、事务和传输，注意它们之间的关系。

1. 域：是 USB 数据最小的单位，由若干位组成，域可分为七个类型：
   1) 同步域（SYNC）：八位，固定值为 0000 0001，用于本地时钟与输入同步；
   2) 标识域（PID）：由四位标识符+四位标识符反码构成，表明包的类型和格式（这是一个很重要的部分，这里可计算出，USB 的标识码有 16 种，具体分类见 Q5）；
   3) 地址域（ADDR）：七位地址，设备在主机上的地址，地址 000 0000 被命名为零地址，是任何一个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 个设备的原因；
   4) 端点域（ENDP）：四位，一个 USB 设备有的端点数最大为 16 个；
   5) 帧号域（FRAM）：11 位，每一个帧都有一个特定的帧号，帧号域最大容量 0x800，对于同步传输有重要意义（同步传输为四种传输类型之一，见下文）；
   6) 数据域（DATA）：长度 为 0～1023 字节，在不同的传输类型中，数据域的长度各不相同，但必须为整数个字节的长度；
   7) 校验域（CRC）：对令牌包和数据包（对于包的分类见下文）中非 PID 域进行校验的一种方法，CRC 校验在通讯中应用很广泛，是一种很好的校验方法。CRC 码的除法是模 2 运算，不同于十进制中的除法；
2. 包：由域构成的包有四种类型：令牌包、数据包、握手包和特殊包，前三种是重要的包，不同的包的域结构不同。
   1) 令牌包：可分为输入包、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 位的 FRAM+CRC5（五位的校验码）

2) 数据包：分为 DATA0 和 DATA1 包，当 USB 发送数据时，当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，所有的数据包都是 DATA0，格式如下：
   SYNC+PID+0~1023 字节
3) 握手包：结构最为简单，格式如下：
   SYNC+PID
   （上面每种包都有不同类型，USB1.1 共定义了十种包，见 Q5）

3. 事务：分别有 IN、OUT 和 SETUP 三大事务，每种事务都由令牌包、数据包和握手包三个阶段构成（这里用阶段的意思是因为这些包的发送都有一定的时间先后顺序）。事务的三个阶段如下：
   1) 令牌包阶段：启动一个输入、输出或设置的事务；
   2) 数据包阶段：按输入、输出发送相应的数据；
   3) 握手包阶段：返回数据接收情况，在同步传输的 IN 和 OUT 事务中没有这个阶段，这是比较特殊的。

   事务的三种类型如下：
   1) IN 事务：
      a) 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；
      b) 数据包阶段——设备跟据情况会做出三种反应（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）：
         i. 设备端点正常，设备往主机里面发送数据包（DATA0 与 DATA1 交替 ）；
         ii. 设备正忙：无法往主机发送数据包就发送 NAK 无效包，IN 事务提前结束，到了下一个 IN 事务才继续；
         iii. 相应设备端点被禁止，发送错误包 STALL 包，事务也就提前结束了，总线进入空闲状态。
      c) 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。
   2) OUT 事务：
      a) 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；
      b) 数据包阶段——主机箱设备发送数据，DATA0 与 DATA1 交替；
      c) 握手包阶段——设备根据情况会做出三种反应：
         i. 设备端点接收正确，设备向主机返回 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；
         ii. 设备正忙，无法往主机发送数据包就发送 NAK 无效包，通知主机再次发送数据；
         iii. 相应设备端点被禁止，发送错误包 STALL 包，事务提前结束，总线直接进入空闲状态。
   3) SETUP 事务：
      a) 数据包阶段——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；
      b) 数据包阶段——主机向设备发送数据（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，共有 11 条，见

Q7）；

c) 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）。

4. 传输：传输由 OUT、IN、SETPU 事务其中的事务构成，传输有四种类型，中断传输、批量传输、同步传输和控制传输。其中中断传输和批量传输的结构一样，同步传输有最简单的结构，而控制传输是最重要的也是最复杂的传输。
   1) 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
   2) 批量传输：由 OUT 事务和 IN 事务构成，用于大容量数据传输，没有固定的传输速率也不占用带宽，当总线忙时，USB 会优先进行其他类型的数据传输，而暂时停止批量传输
   3) 等时传输：由 OUT 事务和 IN 事务构成，有两个特殊的地方，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是 DATA0
   4) 控制传输：最重要的也是最复杂的传输，控制传输由三个阶段构成（初始设置阶段、可选数据阶段、状态信息步骤），每一个阶段都可以看成一个传输，也就是说控制传输其实是由三个阶段构成的，用来于 USB 设备初次连接到主机之后，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，这是每一个 USB 开发者都要关心的问题。
      a) 初始设置步骤：就是一个由 SETUP 事务构成的传输
      b) 可选数据步骤：就是一个由 IN 或 OUT 事务构成的传输，这个步骤是可选的，要看初始设置步骤有没有要求读/写数据（由 SETUP 事务的数据包阶段发送的标准请求命令决定）
      c) 状态信息步骤：这个步骤就是要获取状态信息，由 IN 或 OUT 事务构成的传输，但要注意这里的 IN 和 OUT 事务与之前的 IN 和 OUT 事务有两点不同：
         i. 传输方向相反，通常 IN 表示设备向主机发送数据，OUT 表示主机向设备发送数据；而在这里，IN 表示主机向设备发送数据，OUT 表示设备向主机发送数据，这是为了和可选数据步骤相结合；
         ii. 在这个步骤里，数据包阶段的数据包都是 0 长度的，即 SYNC+PID+CRC16

思考：这些传输模式在实际操作中应通过什么方式设置？

Q5：标识码有哪些？

标识码由四位数据组成，可以表示十六种标识码。在 USB1.1 规范里，只用了十种标识码，USB2.0 使用了十六种标识码。标识码的作用是用来说明包的属性的，标识码是和数据包联系在一起的。数据包分为令牌包、数据、握手和特殊包四种，标识码分别有以下十六种：

1. 令牌包：
   0x01 输出（OUT）启动一个方向为主机到设备的传输，并包含了设备的地址和标号
   0x09 出入（IN）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
   0x05 帧起始（SOF）表示一个帧的开始，并包含了相应的帧号
   0x0d 设置（SETUP）启动一个控制传输，用于主机对设备的初始化
2. 数据包：
   0x03 偶数据包（DATA0）
   0x0b 奇数据包（DATA1）
3. 握手包：
   0x02 确认接收到无误的数据包（ACK）

0x0a 无效，接收（发送）端正忙而无法接收（发送）信息

0x0e 错误，端点被禁止或不支持控制管道请求

4. 特殊包：

0x0c 前导，用于启动下行端口的低速设备的数据传输

Q6：USB 主机是如何识别 USB 设备的？

当 USB 设备插上主机时，主机就通过一系列的动作来对设备进行枚举配置（配置是属于枚举的一个态，态表示暂时的状态），这些态如下：

1. 接入态（Attached）：设备接入主机后，主机通过检测信号线上的电平变化来发现设备的接入；
2. 供电态（Powered）：就是给设备供电，分为设备接入时的默认供电值和配置阶段后的供电值（按数据中要求的最大值，可通过编程设置）
3. 缺省态（Default）：USB 在被配置前，通过缺省地址 0 与主机进行通信
4. 地址态（Address）：经过配置，USB 设备被复位后，就可以按主机分配给它的唯一地址来与主机通信，这种状态就是地址态
5. 配置态（Configured）：通过各种标准的 USB 请求命令来获取设备的各种信息，并对设备的某些信息进行改变或设置
6. 挂起态（Suspended）：总线在给设备供电 3ms 内没有任何操作，即 USB 总线处于空闲状态的话，该设备就要自动进入挂起状态。在进入挂起状态后，总的电流功耗不超过 280uA

Q7：标准的 USB 设备请求命令究竟是什么？

标准的 USB 设备请求命令是用在控制传输中的“初始设置步骤”里的数据包阶段（即 DATA0，由八个字节构成），标准 USB 设备请求命令共有 11 个，大小都是 8 个字节，具有相同的结构，由 5 个字段构成（字段是标准请求命令的数据部分），结构如下（括号中的数字表示字节数，首字母 bm、b、w 分别表示位图、字节、双字节）：

bmRequestType(1)+bRequest(1)+wvalue(2)+wIndex(2)+wLength(2)

各字段的意义如下：

1. bmRequestType：D7D6D5D4D3D2D1D0

   D7 =0 主机到设备，
   =1 设备到主机

   D6D5 =00 标准请求命令
   =01 类请求命令
   =10 用户定义的命令
   =11 保留值

   D4D3D2D1D0 =00000 接收者为设备
   =00001 接收者为设备
   =00010 接收者为端点
   =00011 接收者为其他接收者
   =其他，其他保留值

2. bRequest：请求命令代码，在标准的 USB 命令中，每一个命令都定义了编号，编号的值就是字段的值，编号与命令名称如下（要注意这里的命令代码与其他字段结合使用，可以说命令代码是标准请求命令代码的核心，正是因为这些命令代码而决定了 11 个 USB 标准请求命令）：

   1) 0 GET_STATUS：用来返回特定接收者的状态
   2) 1 CLEAR_FEATURE：用来清除或禁止接收者的某些特性

3) 3 SET_FEATURE：用来启用或激活命令接收者的某些特性
4) 5 SET_ADDRESS：用来给设备分配地址
5) 6 GET_DESCRIPTOR：用于主机获取设备的特定描述符
6) 7 SET_ DESCRIPTOR：修改设备中有关的描述符，或者增加新的描述符
7) 8 GET_CONGFIGURATION：用于主机获取设备当前的配置值（与上面的不同）
8) 9 SET_CONGFIGURATION：用于主机知识设备采用的要求的配置
9) 10 GET_INTERFACE：用于获取当前某个接口描述符编号
10) 11 SET_INTERFACE：用于主机请求设备用某个描述符来描述接口
11) 12 SYNCH_FRAME：用于设备设置和报告一个端点的同步帧

Q8：在标准 USB 请求命令中，经常会看到 DESCRIPTOR，这是什么？

DESCRIPTOR 即描述符，是一个完整的数据结构，可以通过 C 语言等编程实现，并存储在 USB 设备中，用于描述一个 USB 设备的所有属性，USB 主机通过一系列命令来要求设备发送这些信息的。它的作用就是通过如问答节中的命令操作来给主机传递信息，从而让主机知道设备具有什么功能、属于哪一类设备、要占多少带宽、使用哪类传输方式及数据量的大小，只有主机确定了这些信息之后，设备才能真正开始工作。标准的描述符有 5 种，USB 为这些描述符定义了编号：

1——设备描述符
2——配置描述符
3——字符描述符
4——接口描述符
5——端点描述符

1. 设备描述符

```
struct   _DEVICE_ DESCRIPTOR_STRUCT
{
    BYTE bLenght;              //设备描述符的字节数大小，为 0x12
    BYTE bDESCRIPTORType;      //描述符类型编号，为 0x01
    WORD bcdUSB;               //USB 版本号
    BYTE bDeviceClass;         //USB 分配的设备类代码，0x01~0xfe 为标准设备类，
                               //0xff 为厂商自定义类型
                               //0x00 不是在设备描述符中定义的，如 HID
    BYTE bDeviceSubClass;      //usb 分配的子类代码，同上，值由 USB 规定和分配的
    BYTE bDeviceProtocl;       //USB 分配的设备协议代码，同上
    BYTE bMaxPacketSize0;      //端点 0 的最大包的大小
    WORD idVendor;             //厂商编号
    WORD idProduct;            //产品编号
    WORD bcdDevice;            //设备出厂编号
    BYTE iManufacturer;        //描述厂商字符串的索引
    BYTE iProduct;             //描述产品字符串的索引
    BYTE iSerialNumber;        //描述设备序列号字符串的索引
    BYTE bNumConfiguration;    //可能的配置数量
}
```

2. 配置描述符

```
struct _CONFIGURATION_DESCRIPTOR_STRUCT
{
```

```
    BYTE bLength;                //设备描述符的字节数大小，为 0x12
    BYTE bDESCRIPTORType;        //描述符类型编号，为 0x01
    WORD wTotalLength;           //配置所返回的所有数量的大小
    BYTE bNumInterface;          //此配置所支持的接口数量
    BYTE bConfigurationVale;     //Set_Configuration 命令需要的参数值
    BYTE iConfiguration;         //描述该配置的字符串的索引值
    BYTE bmAttribute;            //供电模式的选择
    BYTE MaxPower;               //设备从总线提取的最大电流
}
```

3. 字符描述符

```
struct _STRING_DESCRIPTOR_STRUCT
{
    BYTE bLength;                //设备描述符的字节数大小，为 0x12
    BYTE bDESCRIPTORType;        //描述符类型编号，为 0x01
    BYTE SomeDESCRIPTOR[36];     //UNICODE 编码的字符串
}
```

4. 接口描述符

```
struct _INTERFACE_DESCRIPTOR_STRUCT
{
    BYTE bLength;                //设备描述符的字节数大小，为 0x12
    BYTE bDESCRIPTORType;        //描述符类型编号，为 0x01
    BYTE bInterfaceNunber;       //接口的编号
    BYTE bAlternateSetting;      //备用的接口描述符编号
    BYTE bNumEndpoints;          //该接口使用端点数，不包括端点 0
    BYTE bInterfaceClass;        //接口类型
    BYTE bInterfaceSubClass;     //接口子类型
    BYTE bInterfaceProtocol;     //接口所遵循的协议
    BYTE iInterface;             //描述该接口的字符串索引值
}
```

5. 端点描述符

```
struct _ENDPOIN_DEscriptOR_STRUCT
{
    BYTE bLength;                //设备描述符的字节数大小，为 0x12
    BYTE bDescriptorType;        //描述符类型编号，为 0x01
    BYTE bEndpointAddress;       //端点地址及输入输出属性
    BYTE bmAttribute;            //端点的传输类型属性
    WORD wMaxPacketSize;         //端点收、发的最大包的大小
    BYTE bInterval;              //主机查询端点的时间间隔
}
```