

UART: 通用异步收发传输器 (Universal Asynchronous Receiver/Transmitter)

- “串口”：这是最通用的说法。虽然串口 (Serial Port) 是一个广义术语，但如果沒有特殊说明，大家默认指的就是 UART
- 波特率的本质是单位时间内传输的“符号 (Symbol)”个数。在 UART 的世界里，一个“符号”就是一个高电平或低电平的电压状态
- **9600** 波特率 的本质含义是：“把 1 秒钟分成 **9600** 份，每一份的时间里，电线上只允许表现一种状态（高或低）。”
- 位宽 (Bit Width) 指的是发送一个 bit 数据在时间上占据的长度。它由波特率 (Baud Rate) 决定，计算公式是：位宽 = 1 / 波特率: 位宽 = $1 / 9600 \approx 104.17 \mu\text{s}$, 这意味着：发送方拉低或拉高电平后，会死掐表，准时等够 $104.17\mu\text{s}$ ，再切换下一个 bit 的电平

波特率 vs 比特率

- 比特率 (**Bit Rate**)：是信息层面的概念。指每秒传了多少个 0、1
- 波特率 (**Baud Rate**)：是物理波形层面的概念。指每秒电平变化了多少次。
- 为什么 **UART** 里它们相等？因为 UART 很原始，1 个电平变化只代表 1 个 bit。
 - 对比：在现代光纤或 WiFi 调制中，通过改变波形的相位和振幅，1 个波形变化（1 波特）可以携带 3 个甚至 10 个 bit。那时候，比特率就是波特率的好几倍。

1. 什么是“位宽”？一个 bit 持续多久？

位宽 (Bit Width) 指的是发送一个 bit 数据在时间上占据的长度。它由波特率 (Baud Rate) 决定，计算公式是：位宽 = 1 / 波特率。

- 举例：如果波特率是 9600 bps (每秒比特数)：
 - 位宽 = $1 / 9600 \approx 104.17 \mu\text{s}$ 。
- 这意味着：发送方拉低或拉高电平后，会死掐表，准时等够 $104.17\mu\text{s}$ ，再切换下一个 bit 的电平。

2. 起始位 (Start Bit) 与 停止位 (Stop Bit) 的作用

由于 UART 没有时钟线，接收方并不知道发送方什么时候会突然发数据。

- 起始位（拉低电平 1 个位时间）：
 - 作用：闹钟/唤醒服务。
 - 操作：UART 规定平时没事 (idle) 就一直是高电平。一旦接收方看到电平从“高”变成“低”，它就知道：“注意！对方要开始说话了！”。
 - 它是 bit 0 吗？在物理电平上，它和 bit 0 是一样的（都是低电平），但在协议逻辑上，它是“信号弹”，不计入数据。
- 停止位（拉高电平）：
 - 作用：强制复位/准备下一次唤醒。
 - 为什么要它？如果你要发送的数据最后一位刚好是 bit 0（低电平），如果没有停止位（强行拉高），线路就会一直维持低电平。当下一个字符到来时，就没有从高到低的“下降沿”来触发起始位了。
 - 所以，停止位保证了线路一定会回到高电平 (idle)，为下一个起始位的“下降沿”留出空间。

3. 每个 bit 什么时候发？如何知道对方收到了？

这是 UART “原始”且“异步”的精髓：

- 什么时候发下一个 bit？
 - 发送方内部有一个极其精确的定时器。发完起始位，等 1 个位时间（比如 $104\mu\text{s}$ ），立即在引脚上输出第一个数据 bit 的电压，再等 1 个位时间，输出第二个……
 - 它完全不管对方收没收到，它是“盲发”。
- 如何知道前一个 bit 被接收了？
 - 答案是：不知道。
 - UART 物理层没有确认机制。发送方发完一个 byte (8 位)，紧接着就可以发下一个。
 - 如果不直接发，对方会少收吗？只要双方约定的波特率是一样的，接收方的表和发送方的表就是同步跑的。接收方检测到起始位后，也会启动定时器，每隔 $104\mu\text{s}$ 准时去采样。

总结：UART 发送一帧数据的“独白”

1. **发送方**：(平时高电平) “我要开始啦！” -> 拉低引脚 (Start Bit)。
2. **发送方**：死等 104.17μs。“现在发第 1 位！” -> 根据数据拉高或拉低。
3. **发送方**：再等 104.17μs。“现在发第 2 位！” ... (重复直到 8 位发完)。
4. **发送方**：再等 104.17μs。“完了，休息一下。” -> 拉高引脚 (Stop Bit)。
5. **接收方**：一直在盯着引脚。“咦，掉到低电平了！肯定要开始了！” -> 启动秒表。
6. **接收方**：等 52μs (半个位宽)，“确认一下，还在低电平吗？在，确定是起始位。”
7. **接收方**：每隔 104μs 准时睁眼瞄一下引脚，“现在是 1，现在是 0...”。
8. **接收方**：存够 8 个位，收工。

这种“只靠约定（波特率）和定时器”而不靠“握手信号”的通信，确实非常原始。但也正是因为它把所有的复杂性都交给了时间管理，才让它变得如此 Solid。

你的直觉非常敏锐：如果让 CPU 去不停地在那儿“盯着”引脚电平看，确实是极大的浪费，这在嵌入式编程里叫“轮询（Polling）”或“位操作（Bit-banging）”。但在现代电子系统中，这件事并不是由 CPU 做的，而是由一个专门的“小助手”——“UART 控制器（硬件外设）”来完成的。

1. 为什么不浪费 CPU？（硬件实现的艺术）

现在的 CPU 内部都有专门的 UART 硬件电路。

- 物理层自动化：你给这个硬件外设设定好波特率（比如 9600），它内部就有一个独立的、非常简单的数字逻辑电路（状态机）。
- 硬件盯梢：CPU 根本不参与采样，这个硬件电路会一直盯着引脚。当它检测到“起始位”并收齐了一个字节（8 个 bit）后，它会把这 8 位拼成一个完整的 Byte，放在一个特定的寄存器里。
- 敲门通知（中断）：只有在收完了一整个 Byte 或者一串 Byte 以后，硬件外设才会给 CPU 发一个“中断（Interrupt）”信号。
 - 打个比方：CPU 是大老板，UART 是前台小秘书。大老板不需要亲自坐在门口看谁送快递，小秘书（硬件）收完快递并拆好包，才按一下内线电话（中断）告诉大老板：“领导，有份文件请过目”。

2. 接收端如何确定“采样时间间隔”？

接收端确定间隔的方法非常死板，但也非常精确：分频器（Divider）。

A. 共享的“共识频率”

虽然 UART 线路没有时钟线，但接收端内部有一个基准时钟（Base Clock）。

- 假设单片机的主时钟是 16MHz。
- 你告诉它：“我要用 9600 的波特率”。

B. 内部自动算账

硬件控制器会自动计算一个 分频数值：\$16,000,000 \div 9600 \approx 1666\$。这意味着，每过 1666 个时钟周期，引脚上就应该走完一个 bit。

C. 过采样的“内部节奏”

为了实现 16 倍过采样，硬件会进一步细分：\$1666 \div 16 \approx 104\$。

- 硬件逻辑：每隔 104 个主时钟周期，我就给引脚“拍个照”（取样一次）。
- 起始位触发计时：
 1. 平时计数器是不动的。
 2. 一旦引脚从 1 变 0（检测到下降沿），计数器瞬间启动。
 3. 数到第 8 个“拍照点”，看一眼。
 4. 之后每隔 16 个“拍照点”，睁眼看一次存起来。
 5. 存够 8 个，停掉计时器，触发中断。

3. 采样频率真的需要“非常高”吗？

其实不需要。

- 9600 波特率：只需要 153.6 kHz (\$9600 \times 16)\$ 的采样脉冲。
- 115200 波特率（常用高速率）：需要约 1.8 MHz 的采样脉冲。

对于现代运行在几十 MHz 甚至几个 GHz 的芯片来说，产生一个几 MHz 的脉冲序列是极其轻松且功耗极低的。

UART、RS232、RS485、Ethernet

你说得对！但在工程领域，这里有一个非常经典的概念混淆，我们需要把它拆解开：UART 是“灵魂”（逻辑协议），而 RS232/RS485 是“肉体”（电平标准）。就像同样是“汉语”（协议），你可以通过大声喊（RS232）传 10 米，也可以通过打手势（RS485）传 1 公里。

1. 核心区别：协议 vs. 物理层

特性	UART	RS-232	RS-485
本质	内核控制器/协议	电气标准（接口）	电气标准（总线）
定义了什么	波特率、起始位、数据位、停止位（即：0 和 1 怎么排队）	电压范围、物理接口（D89）、抗干扰能力	差分信号、半双工、多设备连接
电压逻辑	TTL 电平 (0V/3.3V)	双极性负逻辑 (-12V/+12V)	差分电平 (A-B 产生的压差)
关系	它是大师，产生原始信号	它把 UART 信号增强并反相	它把 UART 信号转为差分信号

2. 为什么大家混着叫?

在日常生活中，大家说“串口”通常指的是物理上的那个接口：

- 老式电脑后的 9 针 D 型口 (DB9)：那是标准的 RS-232 串口。它内部由一个 UART 控制器产生信号，然后通过一个“电平转换芯片”(如 MAX232) 把 3.3V 变成了 ±12V 发送出去。
- 工业控制里的两根线 (A/B 线)：那是 RS-485。它内部同样是一个 UART 在发 0 和 1，只是物理层为了抗干扰，变成了差分信号。
- 单片机 (Arduino/STM32) 上的引脚：那是 TTL 串口。它是最纯粹的 UART，直接出 OV 和 3.3V。如果你直接把这个引脚接到电脑的 RS-232 串口上，芯片会因为电压太高被烧毁。

3. 三者的哲学演进

1. UART (TTL)：原生哲学。最省电、最简单，但由于电压低且共地，它跑不远（超过 1 米就容易因干扰出错）。它是芯片内部通信的基石。

2. RS-232：距离演进。为了让电脑能连几米外的打印机。它把电压拉高到 12V 左右，通过提高“噪声容限”来延长距离。但它依然是两点通信（一对一）。

3. RS-485：抗干扰/组网演进。为了在嘈杂的工厂里传几百米。它不再看单根线的电压，而是看两根线的压差。这不仅能抵消环境干扰，还允许在一根总线上挂载 32 个甚至更多设备。

总结

- UART 是算法和时序（灵魂）；
- RS-232/485 是物理外壳（衣服）；
- 当你用 `jSerialComm` 编程时，你操作的是 UART 逻辑（设置波特率等），至于电线上流的是 3.3V 还是 12V，软件层通常是不感知的。

就像你用手机发微信，微信协议是 UART，而你的手机是用 5G 信号 (RS232) 还是用 WiFi 信号 (RS485) 把这条消息发出去，并不改变“微信协议”本身的逻辑。

对UART、RS232及RS485等与Ethernet有何关系，日常编程中交互是通过ethernet，如何转换

> Thought for 1s

这是一个非常具有实战意义的问题。在现代工业、物联网 (IoT) 或系统调试中，我们经常会遇到“本地是串口，远程是网口”的尴尬局面。

要把 UART/RS232/RS485 与 Ethernet (以太网) 联系起来，我们可以从 **层级关系** 和 **转换方案** 两个角度来理解。

1. 它们之间的层级关系 (公路 vs. 铁路网络)

你可以把它们想象成两套完全不同的交通系统：

- UART / RS232 / RS485 (串口)：私家车/乡村公路。**
 - 特点：**点对点，结构极其简单。没有复杂的站点（地址）、没有列车调度中心（路由器）。只要路（线）连上了，车（数据）就能开过去。
 - 层级：**只有物理层（电压）和极简的数据链路层（起始/停止位）。
- Ethernet (以太网)：国家铁路网。**
 - 特点：**极其复杂。需要地址 (IP)、需要列车编号 (端口/Port)、需要一套庞大的规章制度 (TCP/IP 协议栈) 来保证行李不丢、顺序不错。
 - 层级：**包含物理网线、MAC 地址、IP 协议、TCP 协议等多个层级。

2. 编程中的交互与转换

在日常编程中，如果你的设备在千里之外，而它只有串口，你通常会通过以下两种“桥梁”来转换：

方案 A：硬件协议栈转换 (串口服务器 / DTU)

这是最常用的工业方案。你会买一个叫“串口服务器”的盒子。

- 物理连接：**盒子一头连 RS485 总线，另一头插网线连入交换机。
- 工作逻辑：**这个盒子内部有一个微小的 CPU，它在运行“透明传输 (透传)”程序。
 - 它收到的每一个 Ethernet 包里的“负载数据”，都会原封不动地从 UART 口吐出来；
 - 从 UART 口收到的每一个字节，都会被它打包成 TCP 或 UDP 包发给远端的服务器。

方案 B：编程逻辑的转换

在代码层面，你会发现两种 API 的风格完全不同：

场景	本地串口通信 (UART)	远程网络通信 (Ethernet)
Java 库	<code>jSerialComm</code>	<code>java.net.Socket</code>
打开方式	<code>port.openPort()</code>	<code>new Socket("192.168.1.100", 8080)</code>
参数确定	需要指定 <code>COM3</code> 、波特率 9600	需要指定 <code>IP 地址</code> 、 <code>端口号</code>
读取数据	监听 <code>SerialPortEvent</code>	通过 <code>InputStream.read()</code>

3. 如何在代码中实现“跨级”转换？

如果你需要写一个程序把串口数据发到网上，流程如下：

1. 打开串口：使用 `jSerialComm` 读到 `byte[] data`。
2. 封装网络包：创建一个 TCP Socket。
3. 发送：`socket.getOutputStream().write(data)`。

反过来也一样。这在工程上叫“协议转换网关”。

4. 一个特殊的辅助工具：“虚拟串口 (VSPD) ”

有时候你的上位机程序（比如一个只支持 COM 口的老软件）必须读 COM 口，但设备在远程网口上。这时我们会使用“虚拟串口驱动”：

- 这个驱动会在 Windows 系统里凭空变出一个 `COM10`。
- 软件往 `COM10` 发数据。
- 驱动在驱动层拦截这些数据，通过网络发给远端的串口服务器。
- 本质：实现了一种“串口号走网络管道”的魔术。

总结比较

维度	UART / RS-xxx	Ethernet (TCP/IP)
传输距离	几米 (RS232) 到 1.2 公里 (RS485)	全球（只要有网络）
寻址能力	无或受限 (485 需特定协议)	极其强大 (IP + Port)
容错性	很低，丢了就丢了	很强 (TCP 会自动重传)
编程难度	极低 (配好波特率即可)	较高 (需处理连接断开、重连、粘包等)

一句话总结：串口是“近身肉搏”，直接操作电平跳动；以太网是“远程狙击”，通过复杂的层层包裹实现极高性能和极远距离的通信。两者通过串口服务器（透传）或软件 Socket 实现无缝衔接。

Good ↗