UM2426
用户手册

STM32Cube X-CUBE-CELLULAR
蜂窝连接扩展包入门

引言

本用户手册介绍了STM32Cube中的X-CUBE-CELLULAR蜂窝连接扩展包的内容和使用。

X-CUBE-CELLULAR扩展包可通过蜂窝网络进行连接。网络接入技术取决于所使用的蜂窝调制解调器：2G、3G、LTE Cat M1或NB-IoT(也称为NB1）。蜂窝连接框架公开标准API，以便使用HTTP协议轻松集成云连接器。

STM32Cube的X-CUBE-CELLULAR扩展包提供了一个应用示例，该示例说明了如何使用HTTP协议连接和订阅云服务，以便将设备数据报告给服务器，以及接收来自远程服务器的命令。

X-CUBE-CELLULAR可用于P-L496G-CELL01和P-L496G-CELL02蜂窝网到云端包。每个包均由一个基于STM32L496的探索主机板组成，该主机板通过STMod+连接器连接到附加蜂窝网调制解调器。P-L496G-CELL01包的附加板配有UG96调制解调器（2G / 3G）。P-L496G-CELL02包的附加板配有BG96调制解调器（LTE Cat M / NB-IoT / 2G回落）。

X-CUBE-CELLULAR扩展包的主要功能如下：

- 已准备好的可运行的固件示例，使用2G、3G、LTECatM1或NB-IoT协议支持物联网云应用的
  快速评估和开发
- 菜单与命令行，通过USB ST-LINK上的虚拟COM UART配置与Grovestreams云物联网平台(HTTP)的连接以及蜂窝连接（技术选择、频段、APN等）
- 蜂窝连接
- 报告温度、湿度和压力等值。如果连接了MEMS附加板（X-NUCLEO-IKS01A2），这些值就是真实值，否则为模拟值。
- 网络无线电级别报告

目录

表格索引

图片索引

1 概述

本用户手册介绍了X-CUBE-CELLULAR扩展包及其用途。本手册未对蜂窝网络和蜂窝协议栈进行解释，可通过互联网获得相关信息。

1.1 术语和定义

*表 1*给出了相关的缩略语定义，帮助您更好地理解本文档。

表1. 缩略语列表

术语	定义
API	应用编程接口
APN	接入点名称
BSD	伯克利软件分发
BSP	板级支持包
C2C	蜂窝网络到云端
CID	上下文ID（蜂窝连接的上下文标识符）
COM	蜂窝通信
DC	数据缓存
eUICC	嵌入式UICC（具有远程配置文件功能的UICC）
eSIM	嵌入式SIM
FEEPROM	表示STM32 MCU的嵌入式闪存
HAL	硬件抽象层
HTTP	超文本传输协议
ICMP	互联网消息控制协议
IDE	集成开发环境
IF	接口
IoT	物联网（参见*[4]*）
IPC	处理器间通道
ITM：	指令跟踪模块
LED	发光二极管
M2M	机器到机器
NAT	网络地址转换
NFMC	网络友好管理配置（参见*[4]*）
NIFMAN	网络IF管理器
MNO	移动网络运营商
MVNO	移动虚拟运营商
PDN	分组数据网络

表1. 缩略语列表（续）

术语	定义
PDU	协议数据单元
PLMN	公共陆基移动网
PPP	点对点协议
PPPoSIF	串行IF PPP
PS	分组交换
RAM	随机存取存储器
ROM	只读存储器
RSSI	接收的信号强度指示
RTC	实时时钟
SMS	短消息服务
TCP	传输控制协议
UDP	用户数据报协议
UICC	通用集成电路卡（也称为SIM卡）
URC	非请求结果码

运行在基于Arm®(a) Cortex®-M处理器的STM32 32位微控制器上。

arm

1.2 参考

1. *STM32Cube扩展包开发指南*用户手册(UM2285)
2. *STM32Cube扩展包开发清单*用户手册(UM2312)
3. *适用于STM32L4系列和STM32L4+系列的STM32CubeL4入门*用户手册(UM1860)
4. *GSM协会的物联网设备连接效率指南*(TSG.34/TS.34)

a. Arm是Arm Limited（或其子公司）在美国和/或其他地区的注册商标。

2 关于安全的重要说明

注意： 应用开发人员必须注意安全方面，并通过实施机制来保护用于连接的令牌和密钥。

X-CUBE-CELLULAR扩展包中提供的应用示例未实施此类保护机制。它只提供了一个基本实现，以便理解栈接口。

警告： 仅在连接天线的情况下使用HW。在没有连接天线的情况下，从天线连接器反射到调制解调器RF输出的功率可能损坏调制解调器。

3 服务连接描述

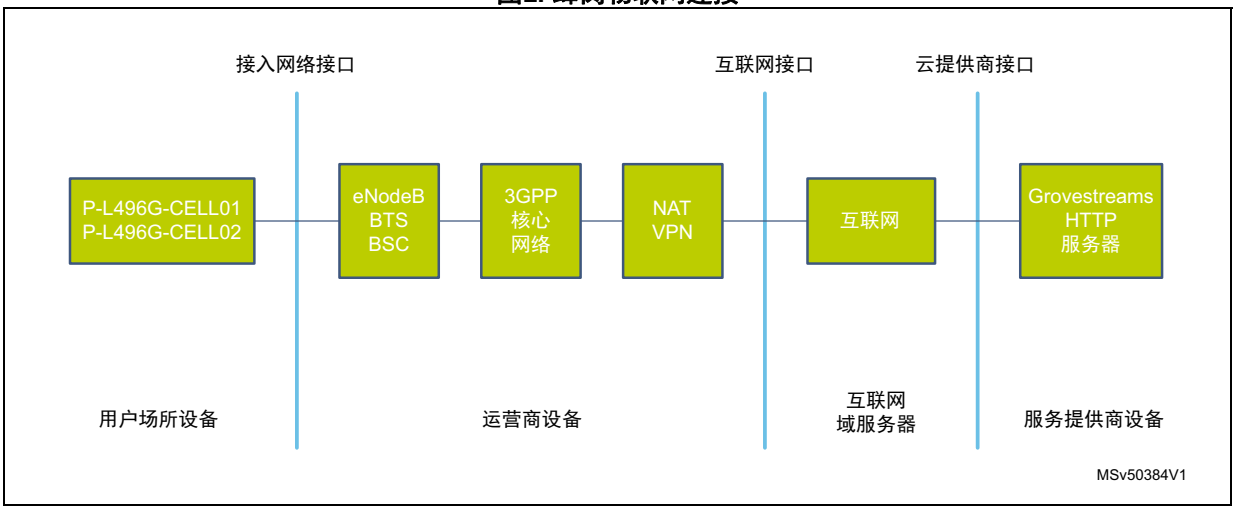
X-CUBE-CELLULAR扩展包提供了通过HTTP协议与互联网进行通信的开箱即用连接。它通过C语言来实现完整的中间件和应用层面栈，从而能够将C2C套件连接到网站。

提供的第一个连接示例连接到Grovestreams网站。在该示例中，主板会向Grovestreams Web浏览器报告通知。

提供的第二个示例实现了ping网络测试功能。

*图 1*显示了由X-CUBE-CELLULAR扩展包处理的蜂窝物联网连接。

图1. 蜂窝物联网连接



蜂窝网络到云端套件包括基于STM32的主板、蜂窝附加调制解调器板以及预付费SIM（从而能够向PLMN注册）。SIM提供商所提供的全球漫游服务使设备在任何国家/地区都能建立连接。SIM仅提供IP连接（意味着不支持SMS）。预付费服务中的可用数据量取决于其使用位置。

通过MNO或MVNO将私有IP地址分配给设备。使用TCP事务请求/响应在设备上运行的任何客户端应用程序都可以通过MNO/MVNO路由器上的IP地址转换(NAT)访问位于互联网中的服务器。

4 包说明

本章详述了X-CUBE-CELLULAR扩展包的内容和使用。

4.1 概述

X-CUBE-CELLULAR扩展包仅提供在主机STM32 MCU上运行的软件组件。蜂窝调制解调器固件不在本文档的范围内。

可支持以下集成开发环境：

- IAR Embedded Workbench® for Arm® (EWARM)
- Keil®微控制器开发套件（MDK-ARM）
- 用于STM32的系统工作台（被称为SW4STM32）

注： *有关支持的IDE版本的信息，请参见交付包的根文件夹中提供的版本说明。*

包中提供了UG96和BG96的IAR™二进制文件。

4.2 调制解调器套接字与LwIP

调制解调器套接字或LwIP均可用于IP栈：

- 调制解调器套接字：在调制解调器FW中运行的IP栈
- LwIP：在STM32端运行的LwIP栈

通过编译过程中使用的标志选择此选项。生成的FW用于调制解调器套接字或LwIP。无法通过启动菜单进一步更改此设置。

注： *如果使用调制解调器套接字，则本用户手册中描述的软件将数据平面支持限制为TCP IPv4客户端应用程序。尚不支持TCP服务器模式和UDP（服务器和客户端）。*

如果使用LwIP，则完全支持TCP和UDP（服务器和客户端）。

如果选择LwIP，则通过PPP层完成主机和调制解调器之间的通信。主机端有PPP客户端，调制解调器端有PPP服务器。PPPoSIF适配LwIP到串行IF，而LwIP通常使用以太网接口。

4.3 架构

X-CUBE-CELLULAR在STM32板上运行，并可以通过附加蜂窝网模块向或从互联网发送或接收IP数据包。

注： *X-CUBE-CELLULAR的某些部分可在裸操作系统环境中使用。完整栈仅与FreeRTOS™配套运行。*

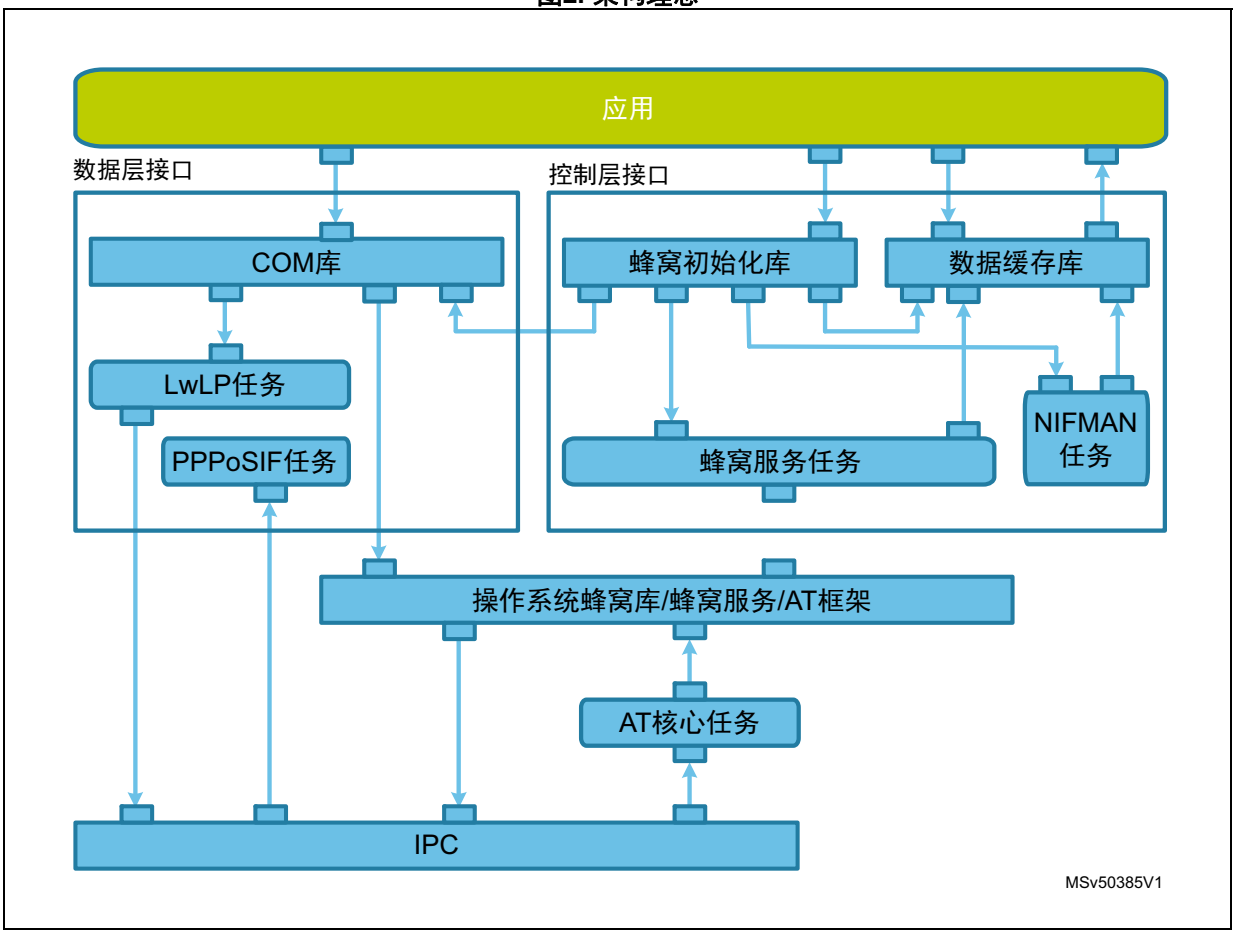
该包分为以下组件：

- STM32L4系列（HAL）
- CMSIS/FreeRTOS™
- LwIP
- AT服务
- 蜂窝服务
- 数据缓存
- IPC
- NIFMAN
- COM
- 蜂窝初始化
- 实用程序

4.3.1 架构理念

本节提供了支持蜂窝网连接的软件架构的高级视图，如*图 2*中所示。

图2. 架构理念



蜂窝连接栈为应用提供了两个主要接口：

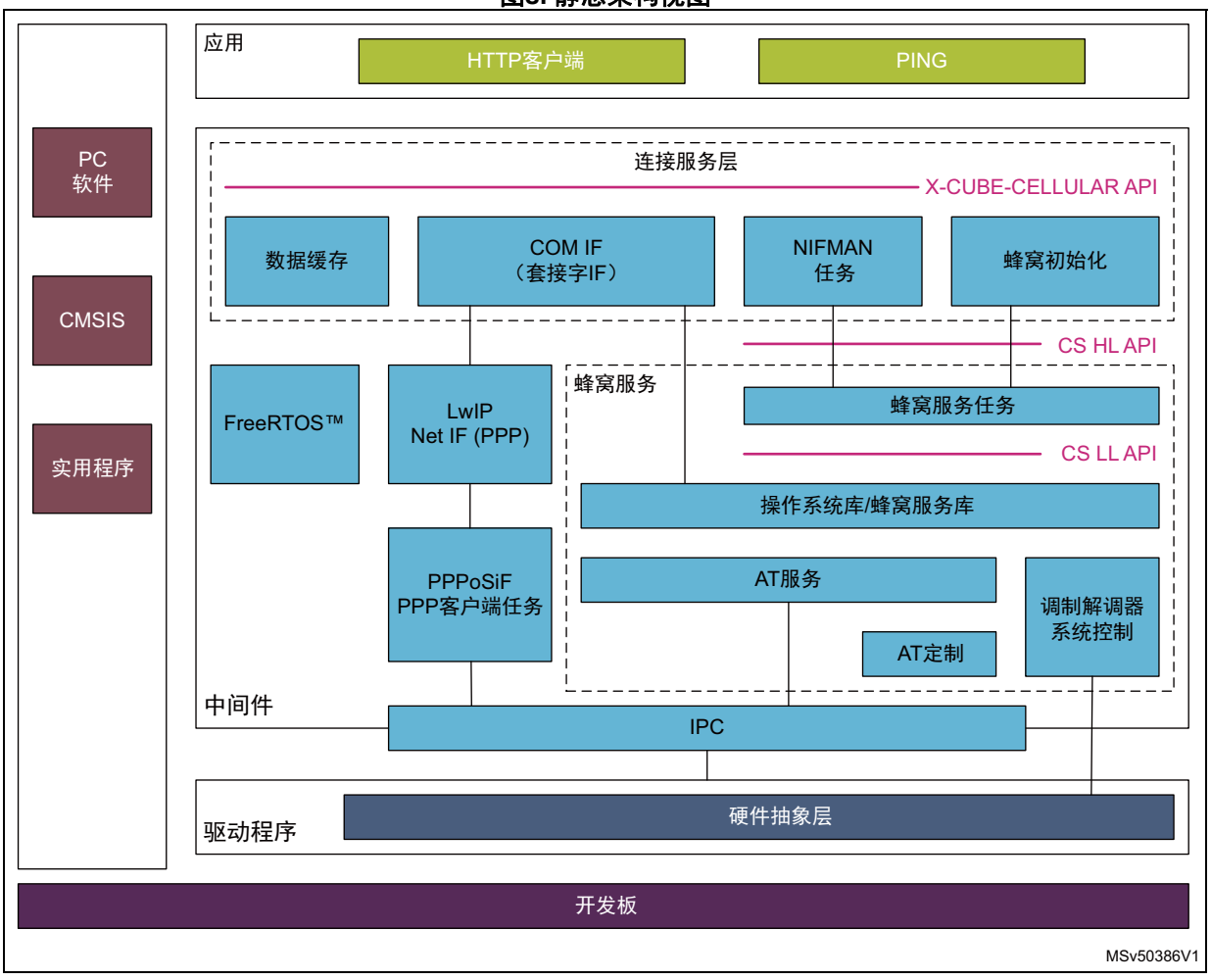
- 控制层接口：有两个控制接口。蜂窝初始化库通过提供API来初始化SW组件和启动蜂窝服务。数据缓存接口用于读取与蜂窝网有关的信息（如信号强度 (RSSI)）和获取事件通知（如网络注册状态变化和网络接口就绪情况）。
- 数据层接口：也称为COM接口，用于向和从远程客户端或服务器发送和接收TCP或UDP段。该接口基于标准BSD套接字API，以便简化应用集成。

IPC层将与调制解调器一起使用的实际HW总线接口抽象化。IPC支持两个逻辑信道，每个信道由一个Tx（到IPC）和一个Rx（从IPC）组成。一个用于与调制解调器交换AT命令，而另一个用于在使用LwIP时携带PPP帧。通过蜂窝服务来控制被选择有的活动通道。

4.3.2 静态架构视图

X-CUBE-CELLULAR静态架构如*图 3*中所示。

图3. 静态架构视图



- **HTTP客户端**：实现HTTP客户端，该客户端将请求发送至云端：www.grovestreams.com。HTTP客户端使用数据缓存来监测网络接口状态变化（从NIFMAN）以及通过TCP的COM套接字接口来发送或接收HTTP数据包。当远程HTTP服务器不可访问时，HTTP客户端还可以实现GSMA TS 34所定义的恢复。
- **连接服务层**：
  - **数据缓存**：对生产者和消费者数据（资源）管理进行解耦的框架。生产者更新的任务资源状态均被推送到数据缓存中（RAM中），这反过来通知最终消费者通过消费应用所提供的回调来处理更新的资源状态。蜂窝服务任务所使用的数据缓存用于发布*蜂窝网络信息，如RSSI*。NIFMAN还使用它来发布网络接口就绪情况。
  - **COMIF**：提供类似BSD的套接字函数集合的库，用于打开、配置以及发送或接收远程TCP或UDP应用的应用PDU。
  - **NIFMAN**：控制网络接口激活的网络接口管理器任务。使用LwIP时，NIFMAN监测PPP服务器状态（调制解调器端），并相应地启动或停止PPP客户端。然后，应用可以在打开用于数据传输的套接字之前监测网络接口状态。
  - **蜂窝初始化库**：公开一个基本功能，以初始化和启动蜂窝服务组件。
- **PPP客户端任务/PPPoSIF**：可选组件。仅在使用LwIP时才存在。负责与调制解调器建立PPP链路。
- **LwIP / Net IF**：LwIP组件及其对PPP的适配
- **蜂窝服务**：
  - **蜂窝服务任务**：控制调制解调器上电和初始化，命令调制解调器执行网络注册，激活PDN（PDP上下文），并进入数据传输模式。通知NIFMAN设置网络接口（PPP链路）。使用AT服务向调制解调器发送AT命令。

    实现了一个通用的有限状态机，以根据调制解调器内部状态变化事件（如FOTA或复位）、网络注册状态变化事件以及与PDP上下文状态相关的事件来维持一致的服务状态。实现*[4]*中所定义的网络友好功能(NFMC)。例如，当错误的APN导致PDP激活失败时，蜂窝服务任务在回落定时器到期后执行新的尝试。

    蜂窝服务任务将蜂窝配置和网络访问参数存储到闪存中，并根据需要配置调制解调器。配置示例包括APN和CID设置、启用和禁用NFMC，以及设置回落定时器。

    为确保系统稳健性，蜂窝服务任务通过定期轮询调制解调器RSSI来确保调制解调器始终在运行。
  - **蜂窝服务操作系统**：是提供一组低级蜂窝服务功能的库。该库串行化了用于与调制解调器通信的单个AT通道接口的访问。这些函数被COM服务和蜂窝服务任务调用。

  - **蜂窝服务库**：提供了一组阻塞函数调用，以便与调制解调器交互。蜂窝服务负责将来自蜂窝服务任务或COM服务的请求转换为必须发送至调制解调器的一系列AT命令。当从调制解调器收到异步事件(URC)时，它最终调用回调函数（来自蜂窝服务任务或COM服务）。
  - **AT服务**：提供了一个框架，以通过IPC向或从调制解调器发送或接收AT命令。AT内核的任务为负责处理蜂窝服务请求并将其转换为AT命令。它还负责处理调制解调器的AT命令响应和URC，并将其转发到蜂窝服务。At分为两部分：

    1. 通用部分，“核心”（AT框架和管理标准AT命令）
    2. 特定部分，“自定义”（实现特定的调制解调器行为和AT命令）
  - **调制解调器系统控制**：支持调制解调器HW系统控制信号（上电/断电、复位）。分为通用和特定部分。通用部分向应用（蜂窝服务）提供通用API，而特定部门则控制专用于调制解调器的GPIO。

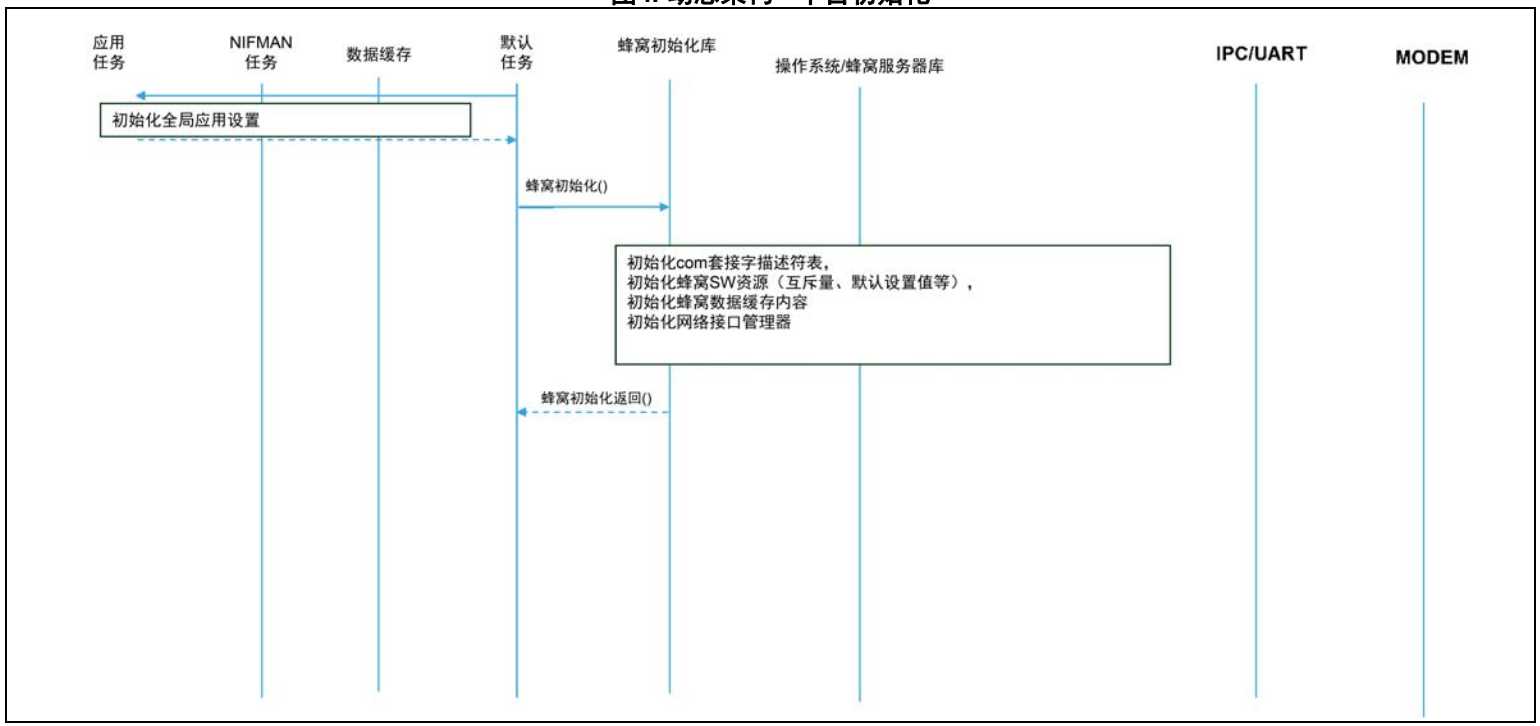
- **IPC**：将实际物理接口(UART)抽象到上层。支持映射到物理信道的逻辑信道处理程序(FIFO)。支持两个信道：字符模式和Stream模式:
  - Stream模式用于数据传输(PPP)。
  - 字符模式用于发送AT命令。
- **实用程序**：提供调试和跟踪等工具。还提供用于更改默认配置的设置菜单（通过串行接口在任何终端上），这在编译和映像创建期间采用硬编码。
- **FreeRTOS™（和CMSIS）**：提供RTOS服务，以创建软件运行所需的资源和调度程序，如线程和任务、动态内存分配、互斥量和信号量。默认任务(*freertos.c*)负责系统初始化和创建所有应用程序任务。最终通过调用cellular_init()和cellular_start()来初始化和启动蜂窝服务组件。

4.3.3 动态架构视图

X-CUBE-CELLULAR动态架构以序列图进一步呈现，以说明选定用例的组件之间的交互：

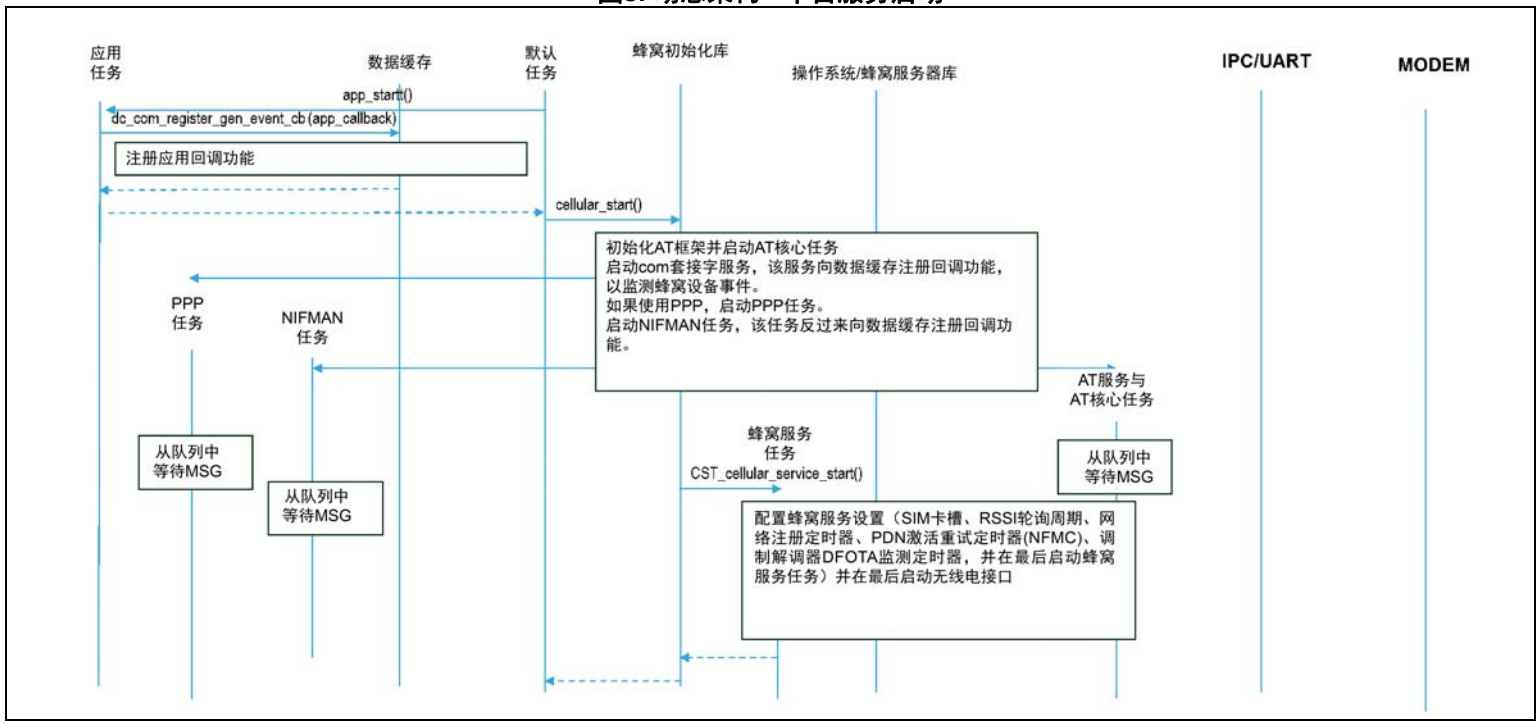
- *图 4：动态架构 - 平台初始化*
- *图 5：动态架构 - 平台服务启动*
- *图 6：动态架构 - 蜂窝服务启动*
- *图 7：动态架构 - 调制解调器初始化*
- *图 8：动态架构 - 蜂窝网络的PLMN搜索和注册*
- *图 9：动态架构 - 分组服务域注册*
- *图 10：动态架构 - PDN激活（调制解调器套接字选项）*
- *图 11：动态架构 - PDN激活（带UG96的LwIP套接字选项）*
- *图 12：动态架构 - PDN激活（带BG96的LwIP套接字选项）*
- *图 13：动态架构 - 套接字创建（调制解调器套接字选项）*
- *图 14：动态架构 - 套接字创建（LwIP套接字选项）*
- *图 15：动态架构 - 数据传输 - 向远程应用发送数据 （调制解调器套接字选项）*
- *图 16：动态架构 - 数据传输 - 向远程应用发送数据 （LwIP套接字选项）*
- *图 17：动态架构 - 数据传输 - 从远程应用接收数据 （调制解调器套接字选项）*
- *图 18：动态架构 - 数据传输 - 从远程应用接收数据 （LwIP套接字选项）*

图4. 动态架构 - 平台初始化



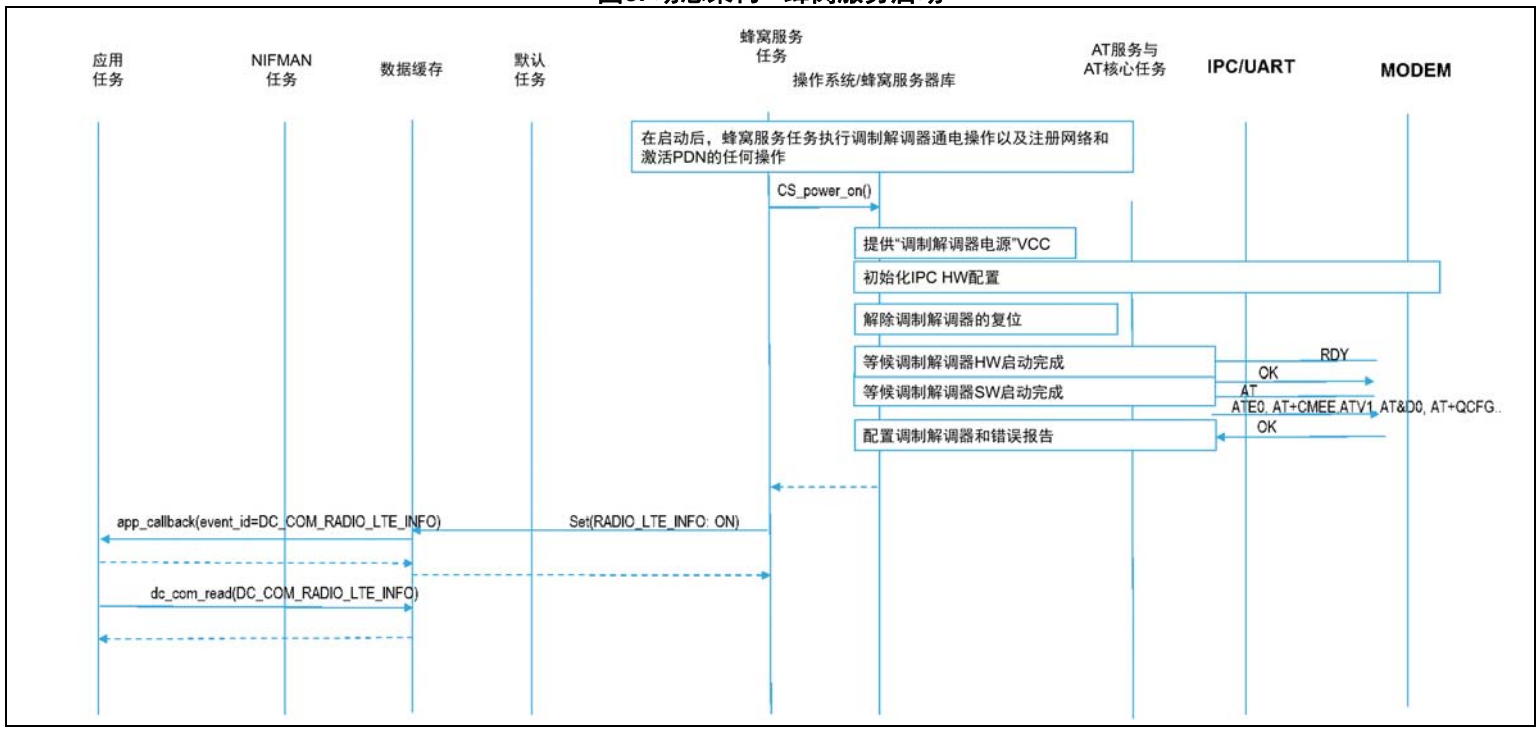
默认任务通过调用cellular_init来初始化所有软件组件和所需的静态资源。

图5. 动态架构 - 平台服务启动



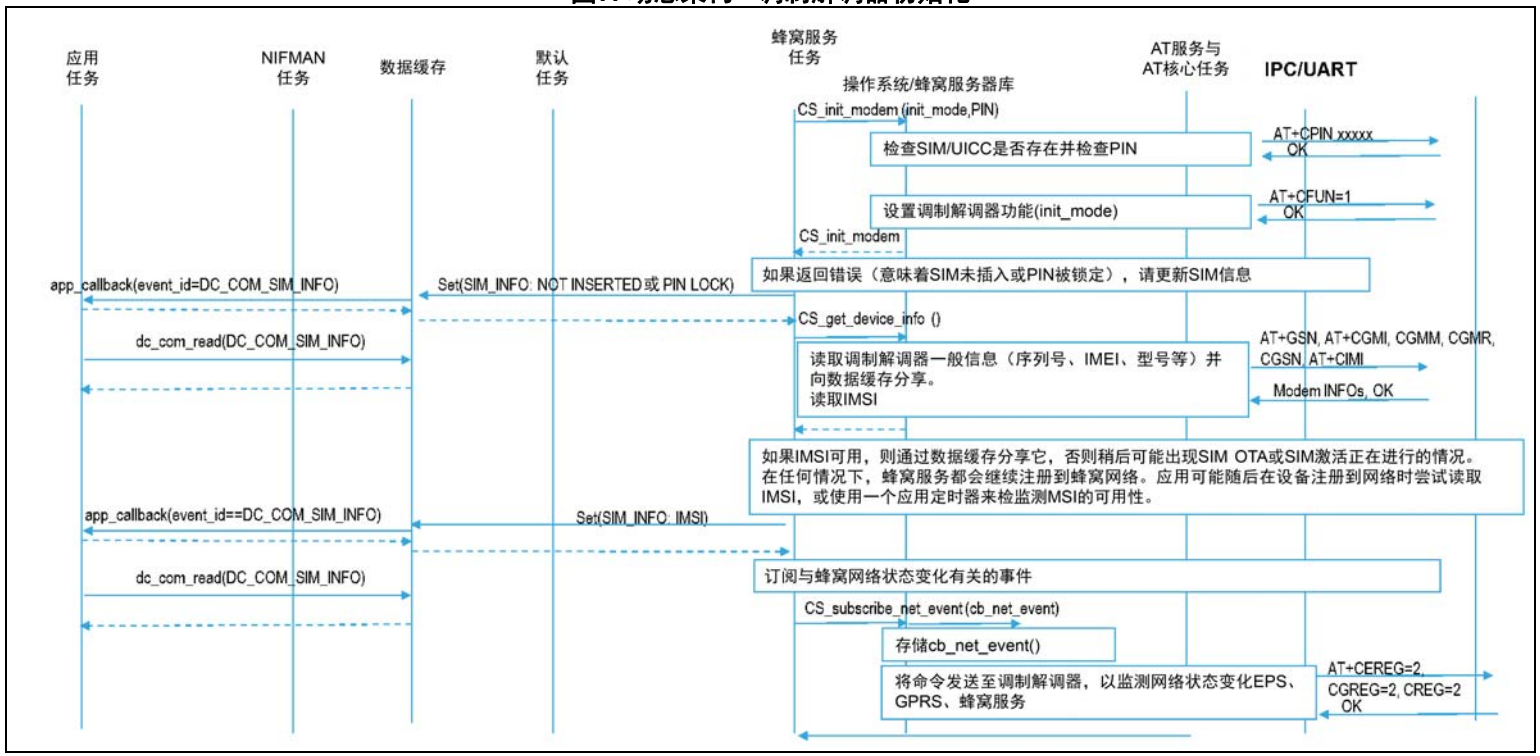
默认任务通过调用 cellular_start()功能来启动软件组件任务和让调制解调器设备上电。

图6. 动态架构 - 蜂窝服务启动



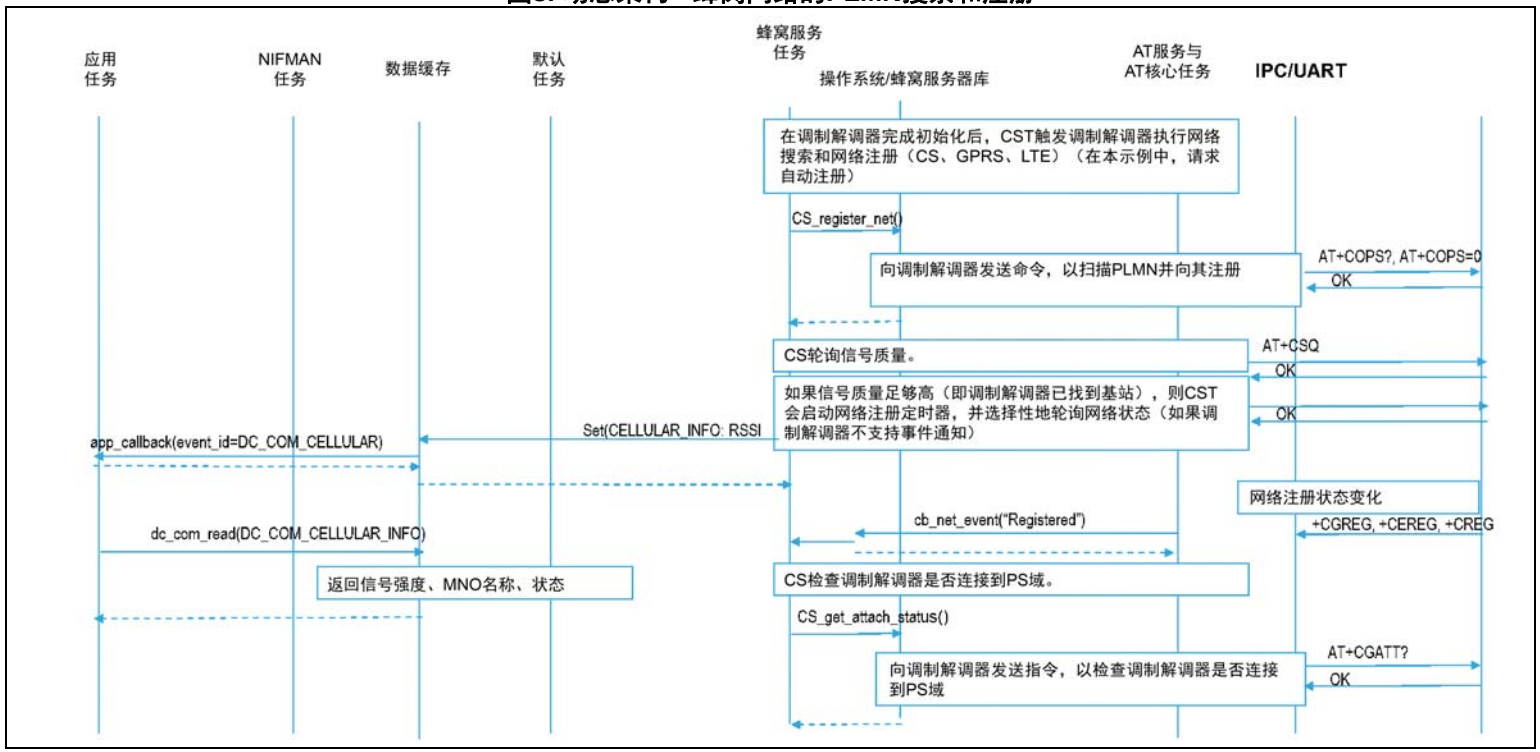
在启动蜂窝服务任务后，将自动执行所有需要的操作，以便让调制解调器设备上电。

图7. 动态架构 - 调制解调器初始化



在调制解调器启动后，蜂窝服务任务会初始化调制解调器功能（完整功能、仅启动）。

图8. 动态架构 - 蜂窝网络的PLMN搜索和注册

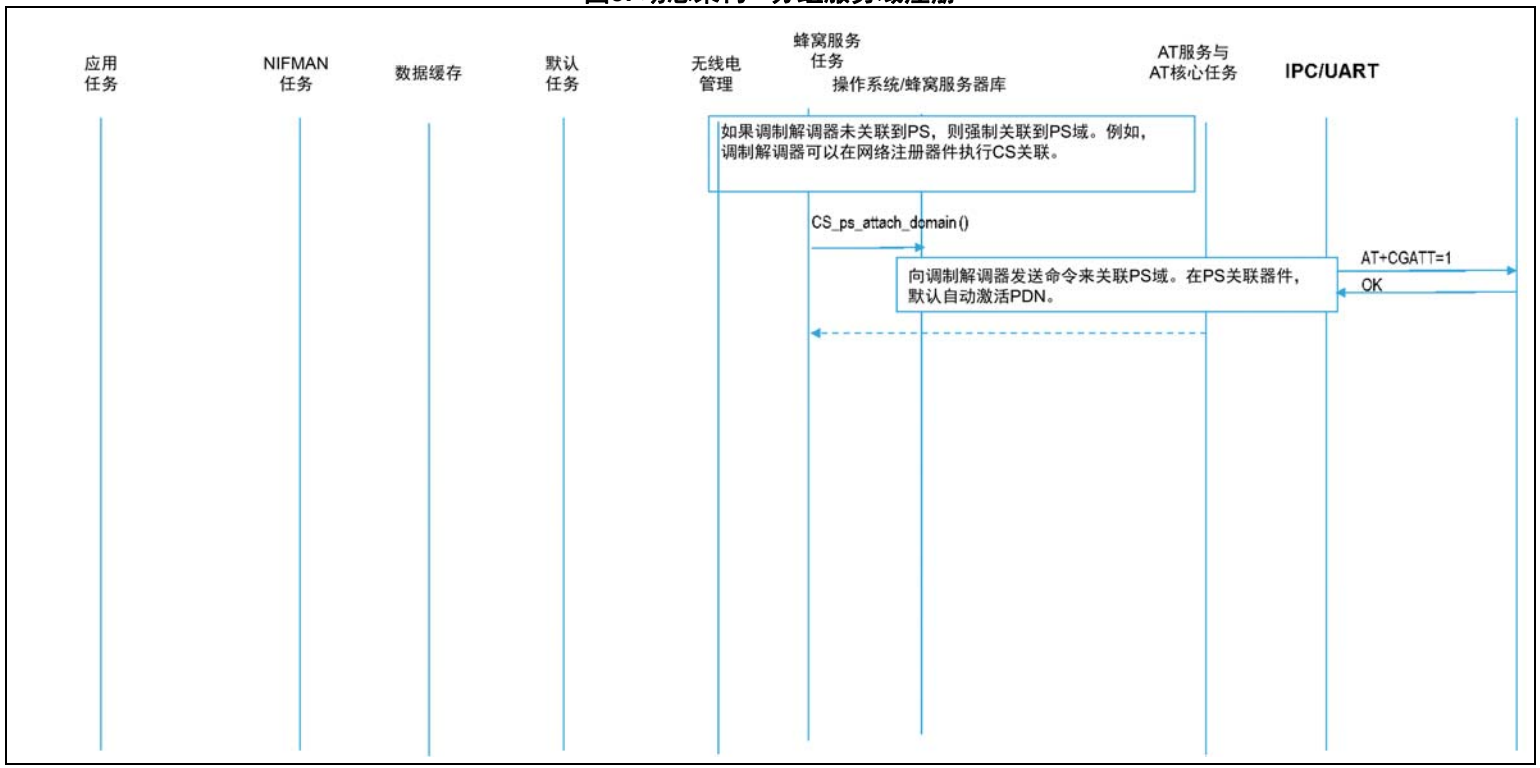


一旦完成调制解调器上下文的初始化，蜂窝服务任务会启动调制解调器，以执行网络搜索（PLMN搜索）并向所选网络注册。在现有实现中，仅支持自动网络搜索。

蜂窝服务会定期轮询信号质量，以检测调制解调器是否已选择一个基站。一旦选择一个基站，蜂窝服务任务就会启动一个定时器，以监测网络注册，因为核心网可能会拒绝注册。如果网络注册定时器超时，蜂窝服务就会指示调制解调器按*[4]*中的定义稍后尝试网络注册（在回落定时器到期后）。

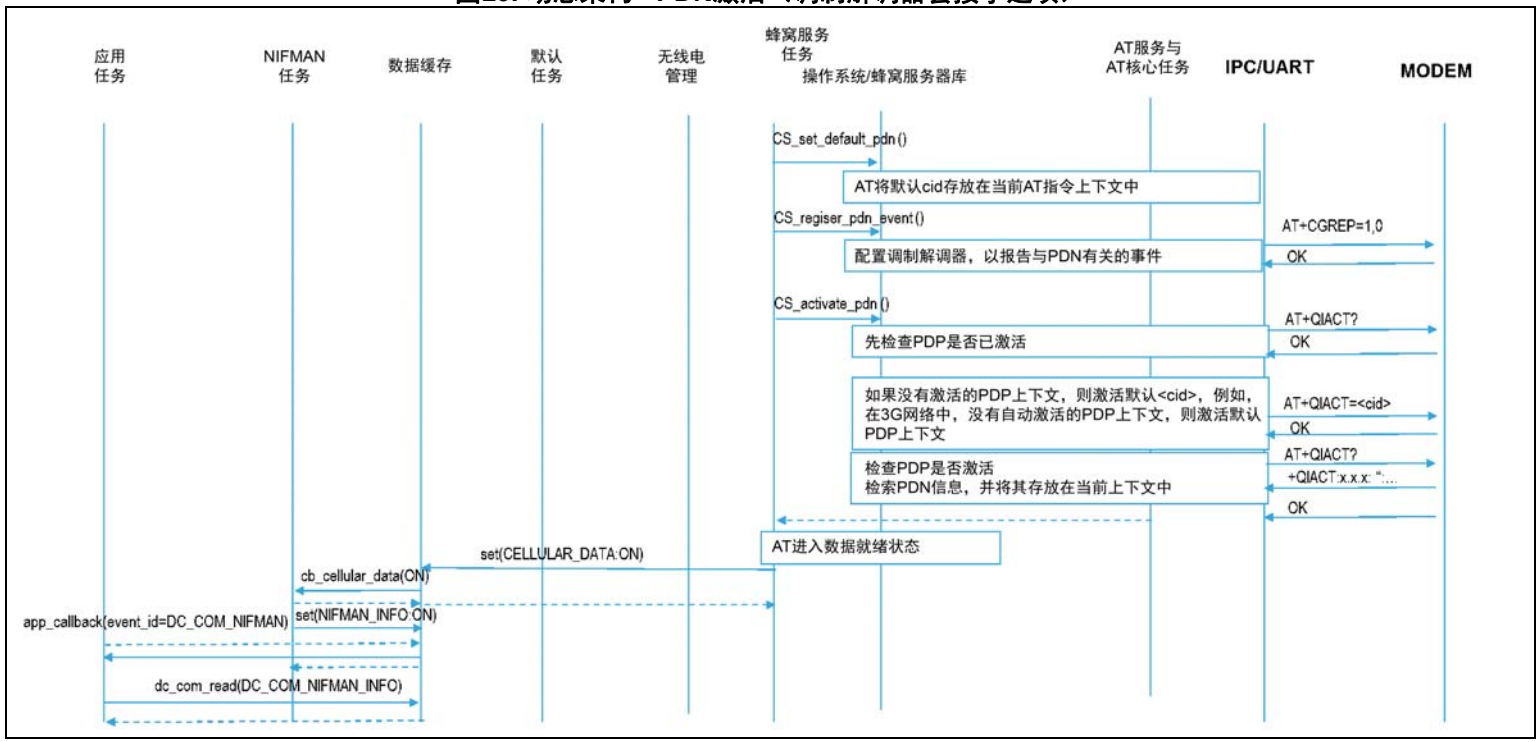
网络注册完成后，蜂窝服务任务验证调制解调器是否已注册到分组域。在2G或3G中，调制解调器可以仅注册到电路业务，但在LTE中，调制解调器在注册时始终连接到分组网络。

图9. 动态架构 - 分组服务域注册



由于目标为运行IP连接，如果调制解调器尚未自动执行分组域注册，则蜂窝服务会强制调制解调器执行分组域注册。

图10. 动态架构 - PDN激活（调制解调器套接字选项）



一旦调制解调器关联到PS域，蜂窝服务任务就会注册回调，以从调制解调器获得与PDN（PDP上下文）相关的通知。它还配置和选择用于数据传输的PDP上下文。

应用可以注册回调（如 app_client_notif_cb()），以获得有关网络接口就绪情况（打开）的通知。

图11. 动态架构 - PDN激活（带UG96的LwIP套接字选项）

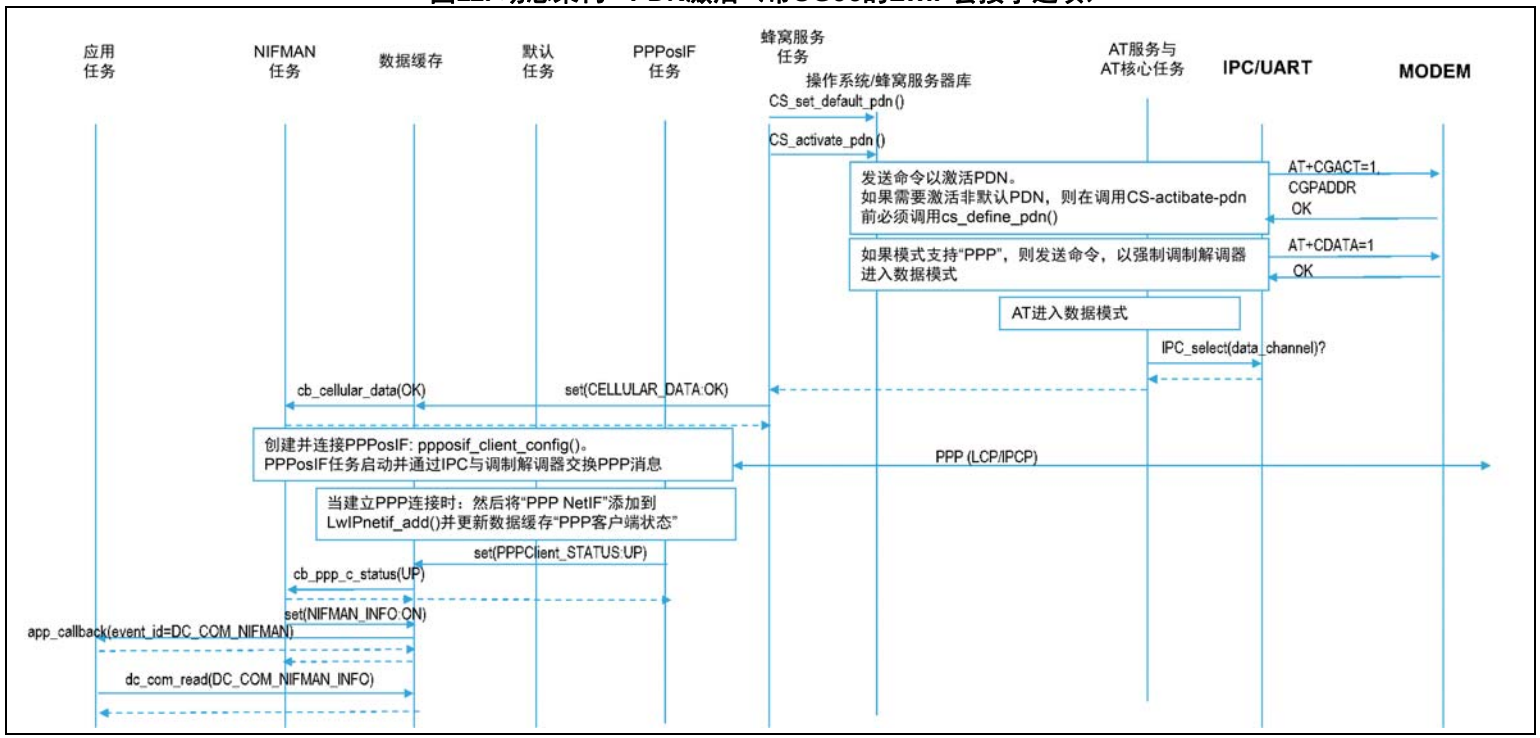
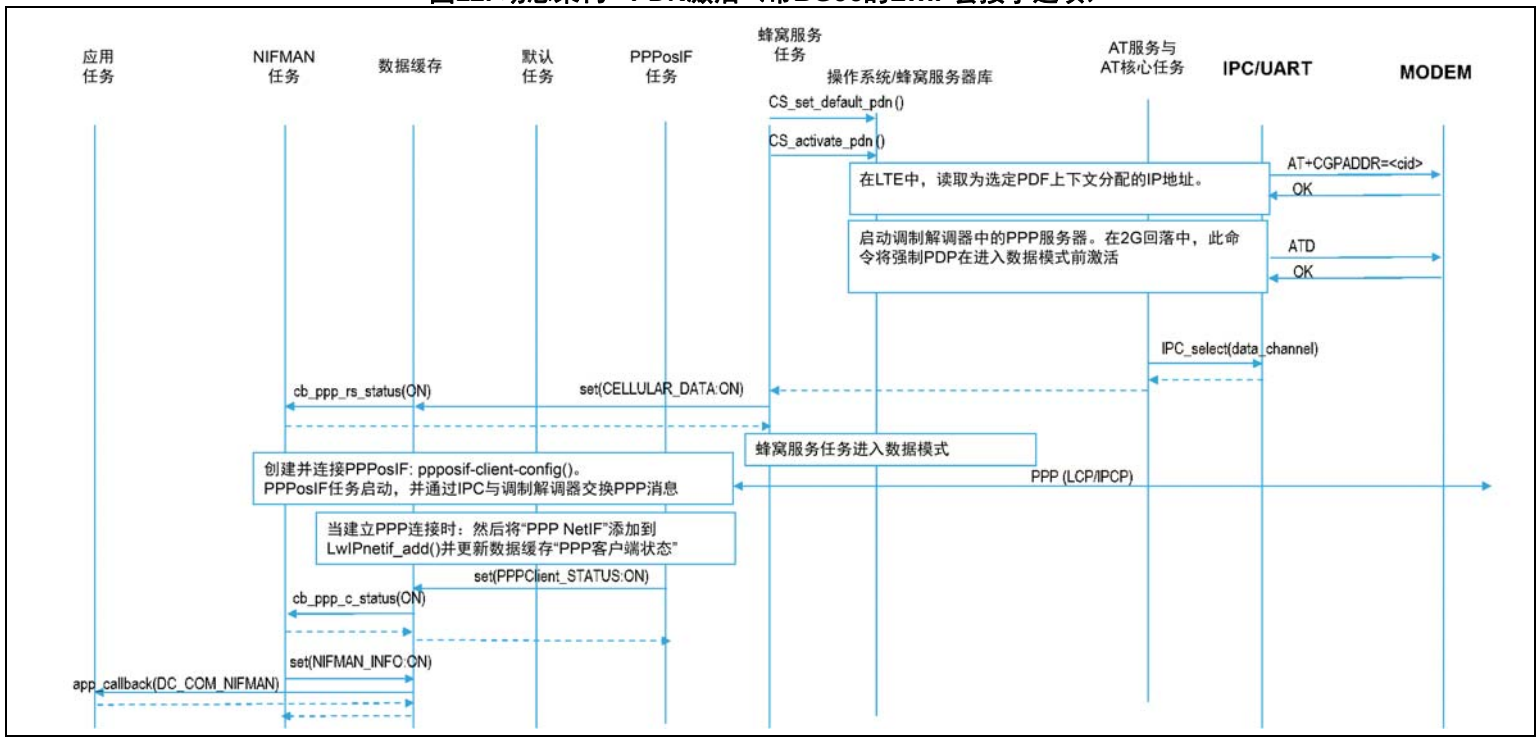


图12. 动态架构 - PDN激活（带BG96的LwIP套接字选项）

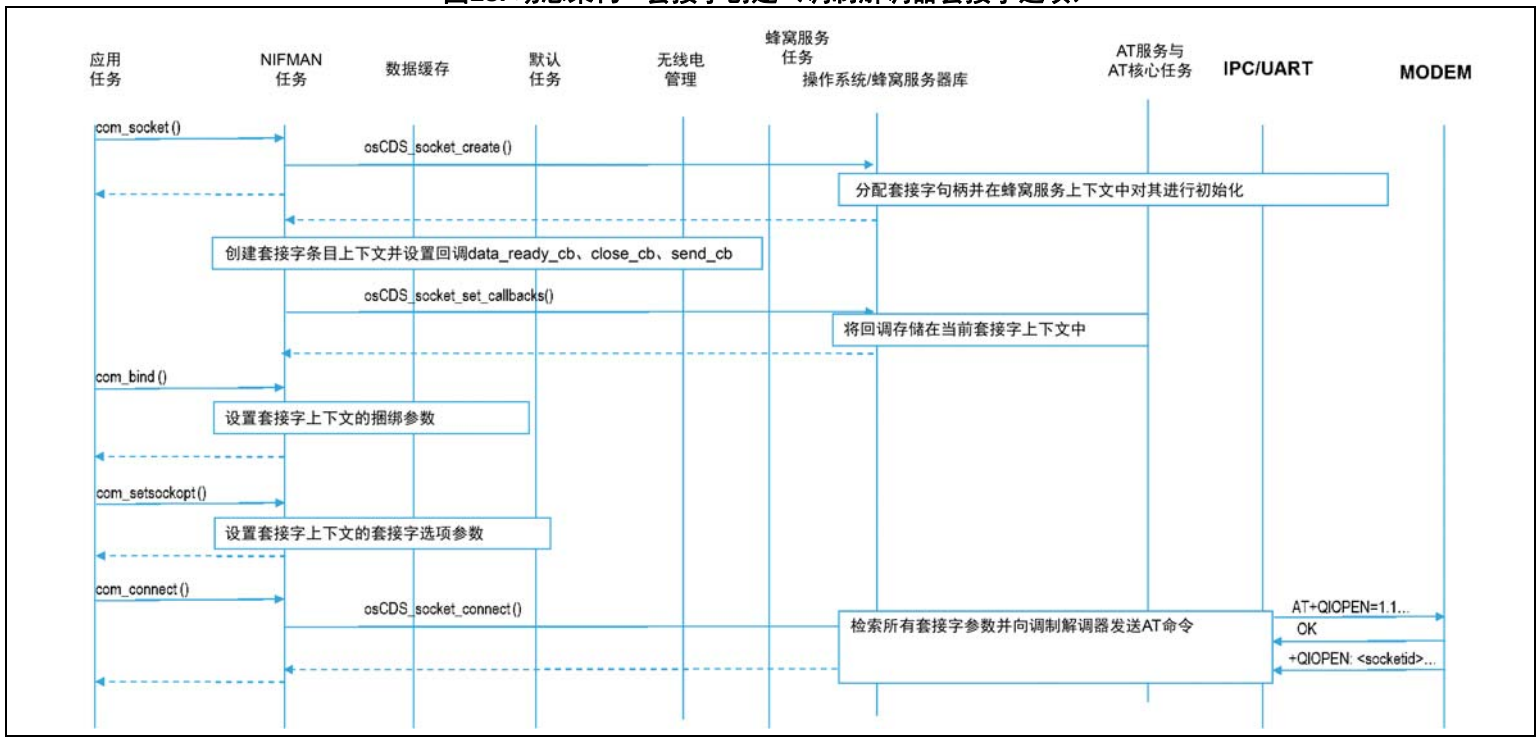


蜂窝服务任务通过调用CS_activate_pdn()函数来发送AT命令，以激活PDP上下文并启动在调制解调器上运行的PPP服务器。如果成功激活PDP上下文和PPP服务器，则蜂窝服务任务通过数据缓存通知NIFMAN。

一旦NIFMAN检测到数据网络接口已准备就绪，它就会命令PPPoSIF任务发起与在调制解调器端运行的PPP服务器的PPP连接。

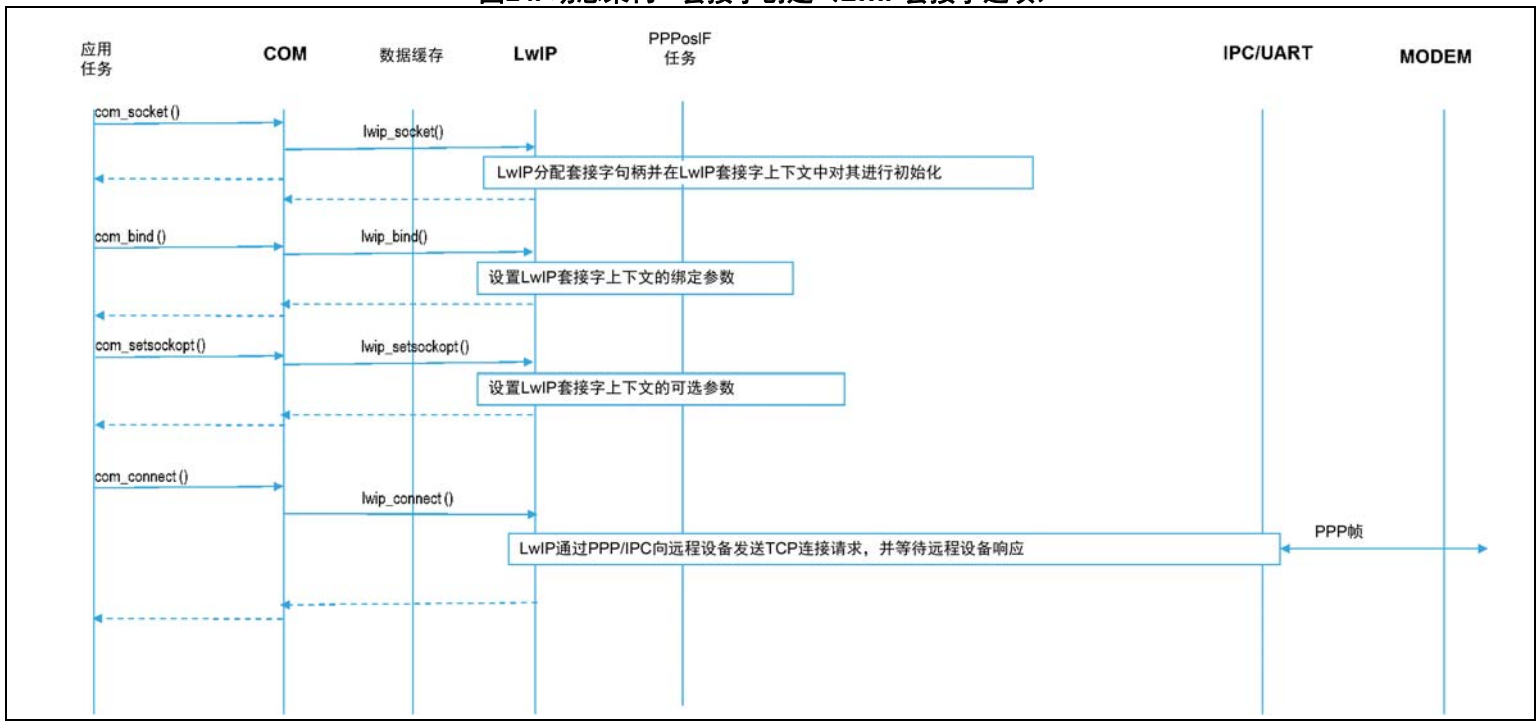
建立PPP连接后，PPPoSIF任务将通过数据缓存通知NIFMAN，并随后通知应用。

图13. 动态架构 - 套接字创建（调制解调器套接字选项）



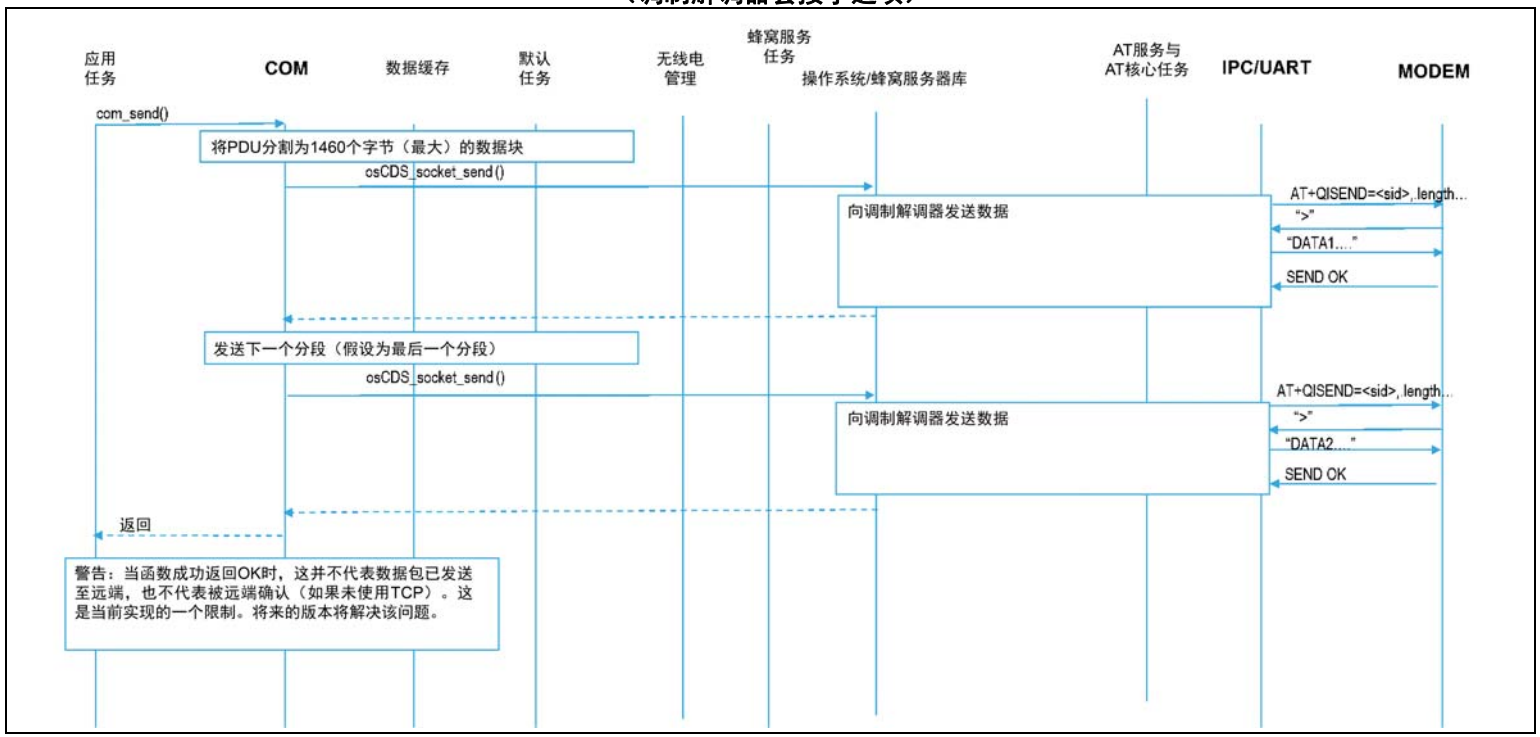
当网络接口准备就绪时，应用可以创建套接字以便与远程应用交换IP数据包。应用调用了com_socket() COM函数，该函数转而调用了蜂窝服务库，以便向调制解调器发送AT命令。

图14. 动态架构 - 套接字创建（LwIP套接字选项）



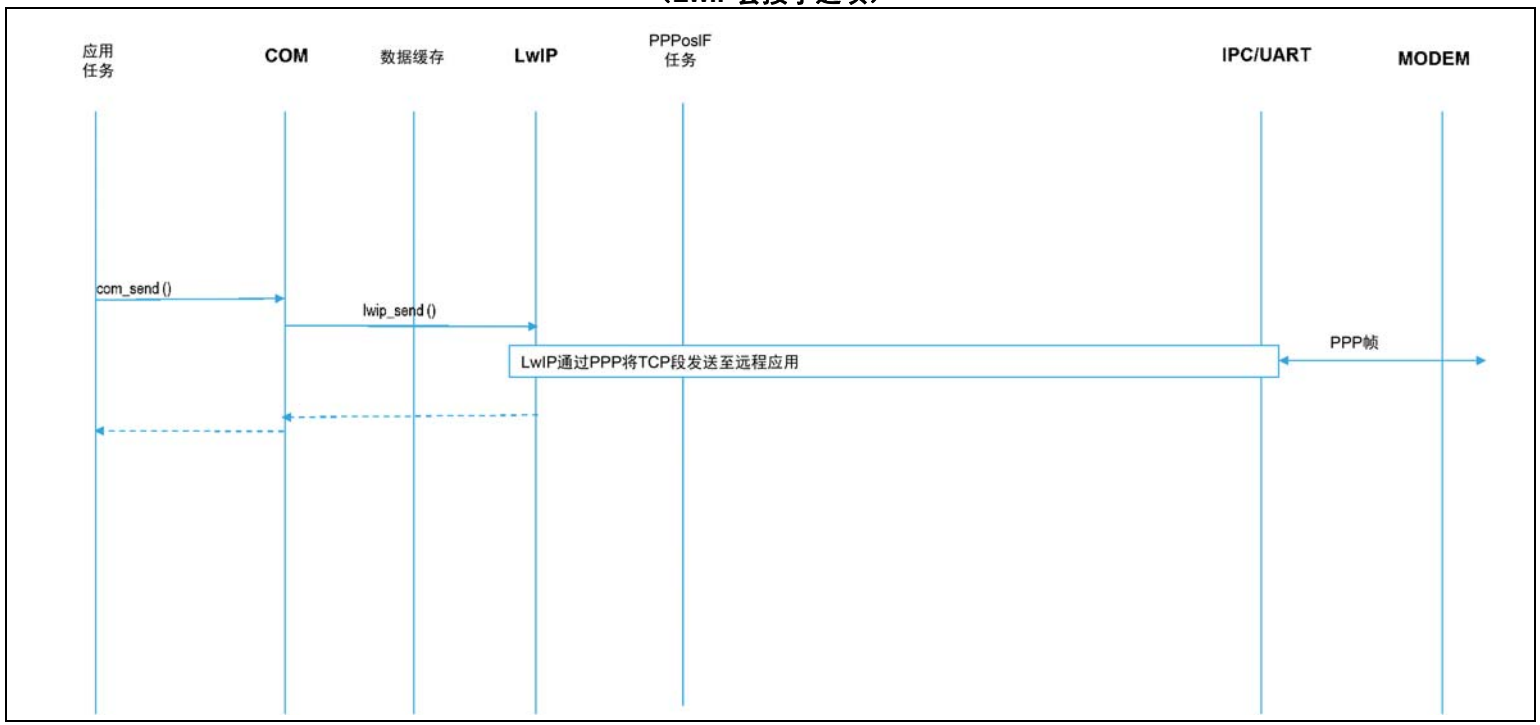
COM接口直接将每个套接字函数映射到LwIP。后者将IP数据包封装到PPP帧中，然后通过IPC层将PPP帧发送到调制解调器。PPPoSIF任务接收来自调制解调器的PPP帧，并将其转发到LwIP栈。

图15. 动态架构 - 数据传输 - 向远程应用发送数据
（调制解调器套接字选项）



COM接口将来自应用的PDU分割成较小的数据块，并通过蜂窝服务库功能将其发送到调制解调器。

图16. 动态架构 - 数据传输 - 向远程应用发送数据（LwIP套接字选项）



com_send()函数调用lwip_send() 函数，该函数由LwIP栈实现，以通过PPP层远程交换TCP段。

图17. 动态架构 - 数据传输 - 从远程应用接收数据
（调制解调器套接字选项）

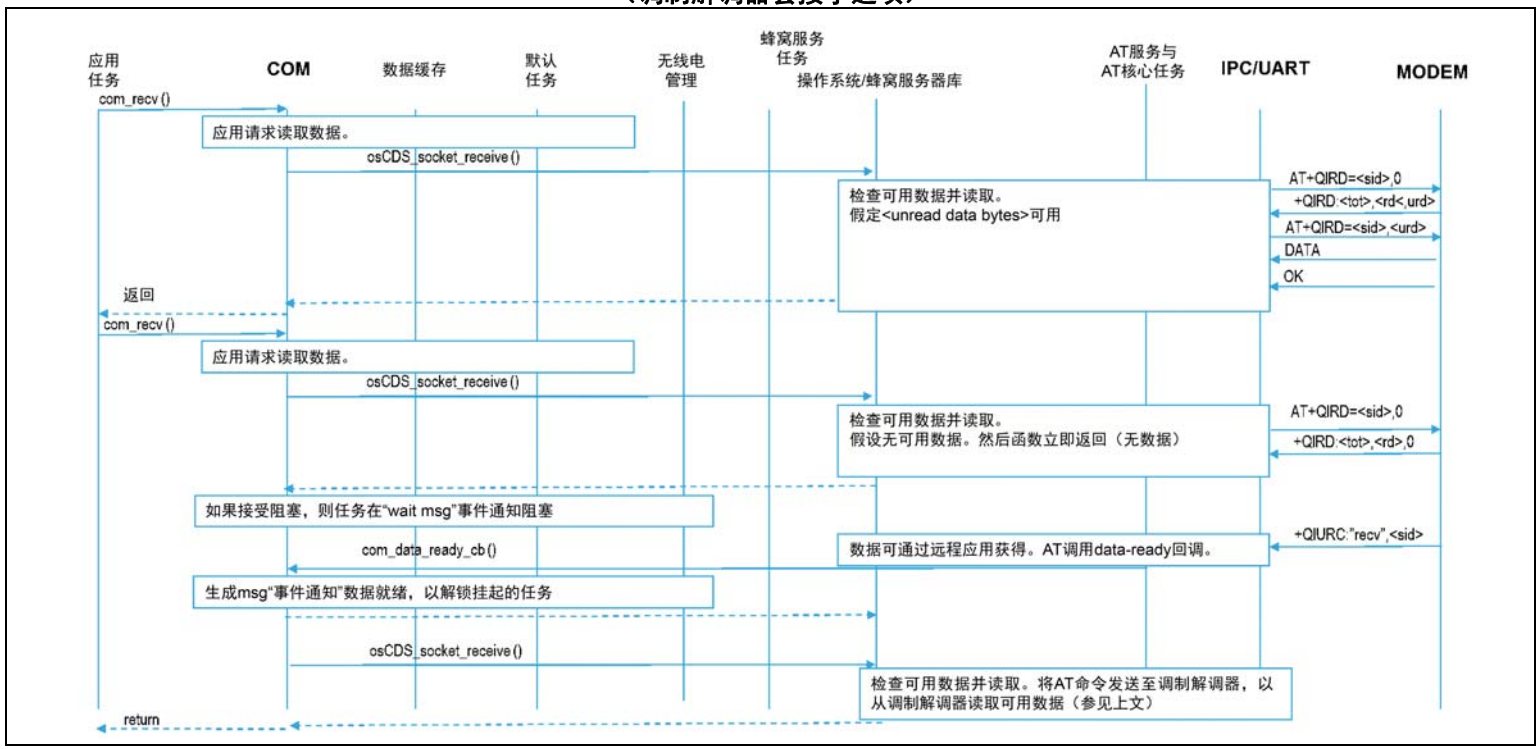
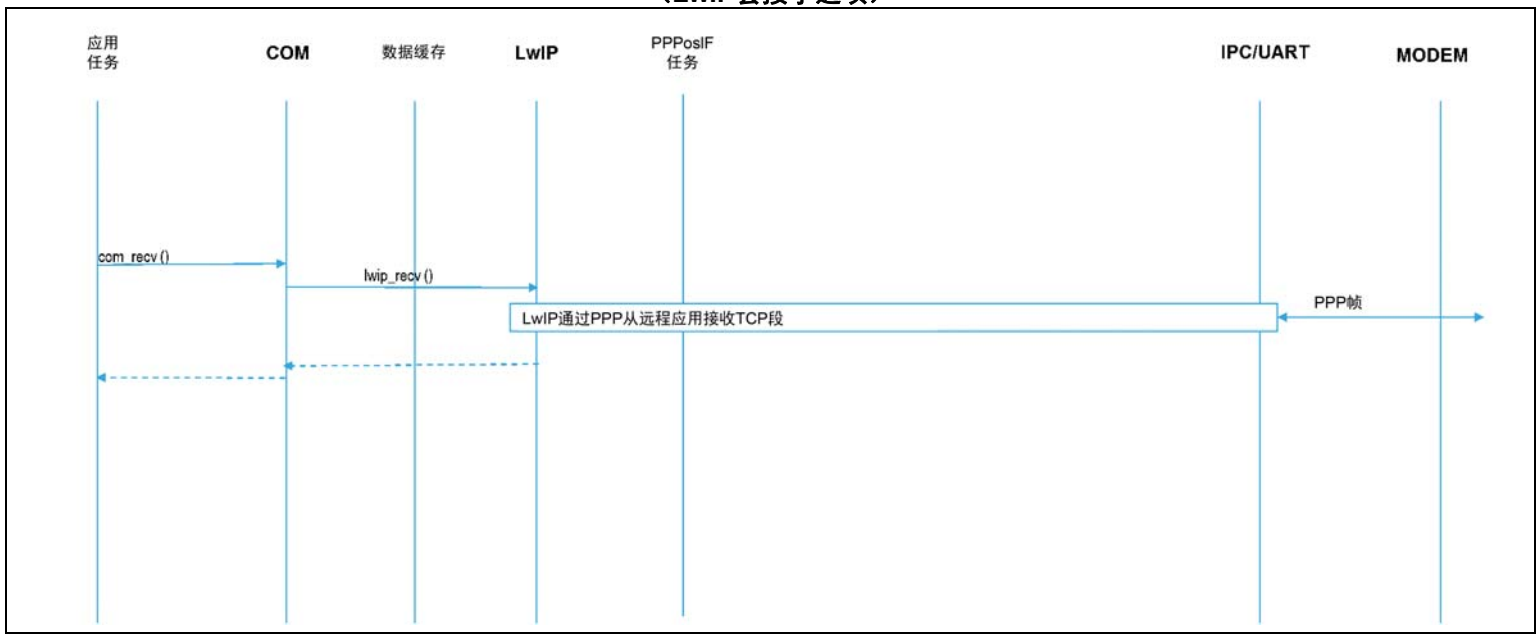


图18. 动态架构 - 数据传输 - 从远程应用接收数据
（LwIP套接字选项）



4.4 X-CUBE-CELLULAR扩展包说明

本节描述X-CUBE-CELLULAR包的软件组成部分。

X-CUBE-CELLULAR是STM32Cube的扩展包。其主要特点为：

- 完全兼容STM32Cube架构
- 为了使应用开发能够访问和使用各种云平台，扩展了STM32Cube。
- 基于STM32微控制器的硬件抽象层STM32CubeHAL

应用程序所使用的软件组件包括：

- **STM32Cube HAL**
  HAL驱动层提供通用的多实例简单API组（应用程序编程接口），以便与上层（应用、库和协议栈）交互。
  它由通用和扩展API构成。它直接围绕通用架构构建，允许在其基础上构建层，例如中间件层，实现了它的功能又无需依赖给定微控制器单元（MCU）的特定硬件配置。
  此结构可提高库代码的可复用性，并确保可向其他设备轻松移植。
- **板级支持包（BSP）**
  除MCU之外，软件包需支持STM32板上的外设。板级支持包（BSP）中包含此软件。这是一个有限的API集，为板特有的某些外设（例如LED和用户按钮等）提供编程接口。
- **应用**
  HTTP和PING客户端。
- **中间件**
  可选LwIP。
- **FreeRTOS™**
  FreeRTOS™是运行X-CUBE-CELLULAR组件任务的必备工具。
- **配置文件**
  通过项目存储库提供组件和平台配置文件
  - *plf_features.h*定义了包含在固件中的功能列表。
  - *plf_hw_config.h*定义了特定于逻辑名称的GPIO和HW接口映射，以便将SW移植到另一个板。它还提供用于与调制解调器通信的HW总线接口配置（如UART）。
  - *plf_sw_config.h*提供了平台SW配置，如任务优先级、跟踪、栈大小监测等。它也提供了应用行为，这些行为可能因平台而异，如调制解调器轮询定时器值、按钮配置和轮询等。
  - *plf_stack_size.h*定义了线程栈大小。

可定制其他参数。*第 8 节：如何定制软件第 51 页*中提供了其他信息。

某些参数也可以在运行时动态定义。它们存储在闪存中，如果需要，在下一次平台启动时可重新使用。

4.5 文件夹结构

*图 19*给出了X-CUBE-CELLULAR包的文件夹结构。

图19. 项目文件结构

4.6 复位按钮

复位按钮（黑色）用于随时复位板子。该操作会强制平台重新启动。

4.7 用户导航键

用户按钮（蓝色）用作导航键：

- 向上：停用或激活对云的重复请求
- 向左：发出ping请求
- 向下：不使用
- 向右：不使用

4.8 应用LED

通过HTTP客户端应用打开或关闭应用的LED。请参见*第 6节：硬件和软件环境设置第 37页*中的*图 23：硬件设置（P-L496G-CELL02示例）*。

4.9 实时时钟

系统日期和时间由RTC管理。启动时，通过HTTP客户端应用更新日期和时间，该应用通过Grovestreams服务器获取时间和日期。

系统日期也可以在启动设置时手动更新。

HAL_RTC_GetTime()函数为应用提供了时间。

注： *小时值取决于Grovestreams服务器时区。*

5 蜂窝连接示例

本章介绍了可用的蜂窝连接示例。提供了几个可并行运行的示例，如PING和HTTP客户端。

5.1 真实网络或模拟器

由于2G和3G网络已覆盖全球，因此能够在真实网络上系统地运行这些技术的示例。LTE Cat M1和NB1技术尚未实现类似的全球覆盖。如果此类网络不可用，用户必须使用符合Cat M1 / NB1标准的网络模拟器。

在本文档中，假设无论使用哪种网络技术，真实网络都可用。

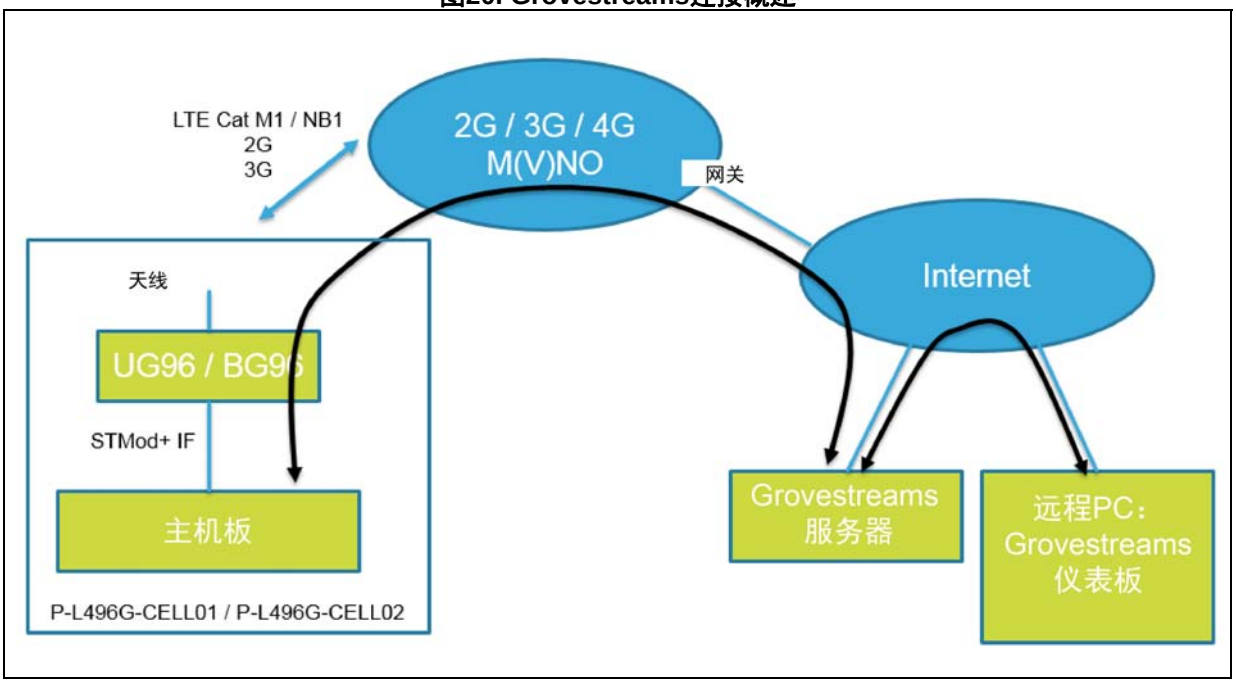
附加调制解调器板嵌入了EMnify MVNO配置文件所提供的eSIM：

- 对于UG96调制解调器，使用2G或3G真实网络
- 对于采用EMnify eSIM的BG96调制解调器，只能使用2G回落。要在Cat M1或NB1模式下使用BG96调制解调器，用户必须插入符合该网络技术标准的塑料SIM卡（由已部署该技术的MNO提供）。

5.2 连接概述

*图 20*中提供了Grovestreams连接概述。

图20. Grovestreams连接概述



5.3 PING示例

只需按下主板左侧的蓝色操纵杆即可运行PING示例。它将PING ICMP请求发送至IP地址为8.8.8.8的Google™，PING应答在跟踪结果中显示。

5.4 Grovestreams (HTTP)访问示例

Grovestreams的蜂窝连接演示包括两个用例：

- 设备定期向Grovestreams云物联网平台报告传感器数据。最终用户连接到Grovestreams Web服务器仪表板。
- 最终用户通过仪表板控制LED应用程序状态。

MCU上的HTTP客户端通过专用帐户建立与Grovestreams服务器的连接。它通过HTTP PUT命令将数据（温度、湿度、压力和蜂窝无线电信号强度）推送到Grovestreams服务器。通过标准Web浏览器连接到Grovestreams平台的最终用户可以访问Grovestreams仪表板的Web页面。通过发送HTTP GET指令来检索最终用户的任何请求，设备每隔两秒对GrovestreamsWeb服务器进行一次轮询。通过这种机制，最终用户可以打开和关闭主机板上的应用LED。

对于温度、湿度和压力，如果插入带传感器的X-NUCLEO-IKS01A2板，则会将实际值发送到云服务器。否则，将报告模拟数据。

*图 21*和*图 22*提供相关的Grovestreams接口。

图21. Grovestreams Web界面，组件视图

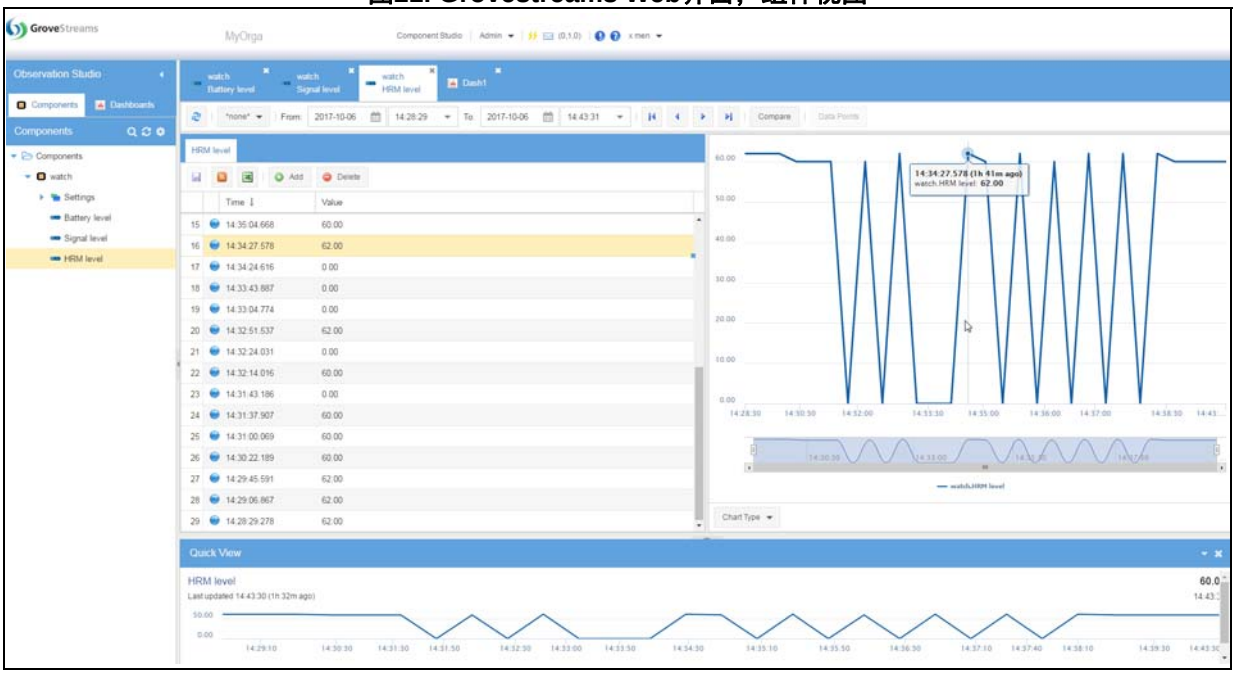
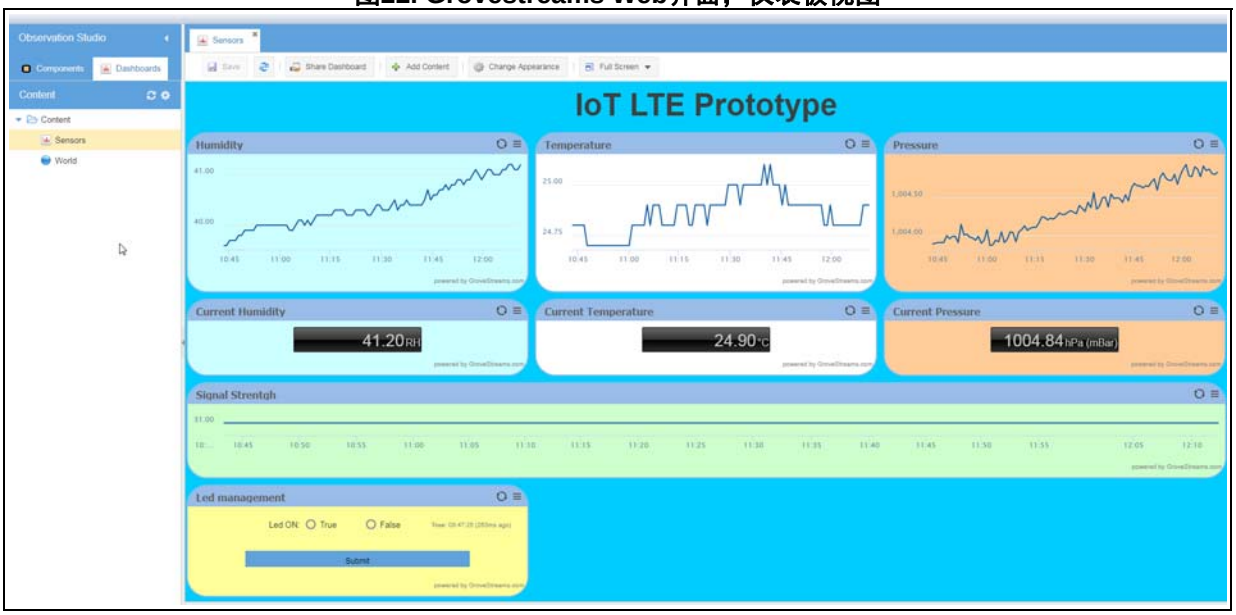


图22. Grovestreams Web界面，仪表板视图



6 硬件和软件环境设置

对于任何关键测试，必须对STM32MCUFW进行编程。如果需要，可升级调制解调器的FW。

注： *在使用新固件进行编程前，使用Teraterm将系统连接到PC并记下凭证编号，需要使用此编号来激活调制解调器板上的eSIM。如果在获得凭证之前烧写板子，请重新烧写原始映像并获取凭证。*

通过STMod +连接器将主机板连接到调制解调器板（请参见*图 24*）。

当未使用焊接到附加板上的UICC芯片时，将符合所用实际网络标准的UICC插入UICC插槽中。UICC插槽位于附加板背面，在USB连接器后面。

如果使用焊接的UICC芯片，则必须事先将其激活，并在启动时通过启动菜单选择该芯片。

通过将其USB连接器插入到PC、USB电源或USB电源组，接通电路板的电源。如果必须显示跟踪结果，则必须将USB连接器连接到带开放控制台应用程序的PC。

警告： **仅在连接天线的情况下使用HW。在没有连接天线的情况下，从天线连接器反射到调制解调器RF输出的功率可能损坏调制解调器。**

*图 23*描述了HW设置，而*图 24*显示了P-L496G-CELL02USB供电线以及可选的跟踪/启动菜单。

图23. 硬件设置（P-L496G-CELL02示例）

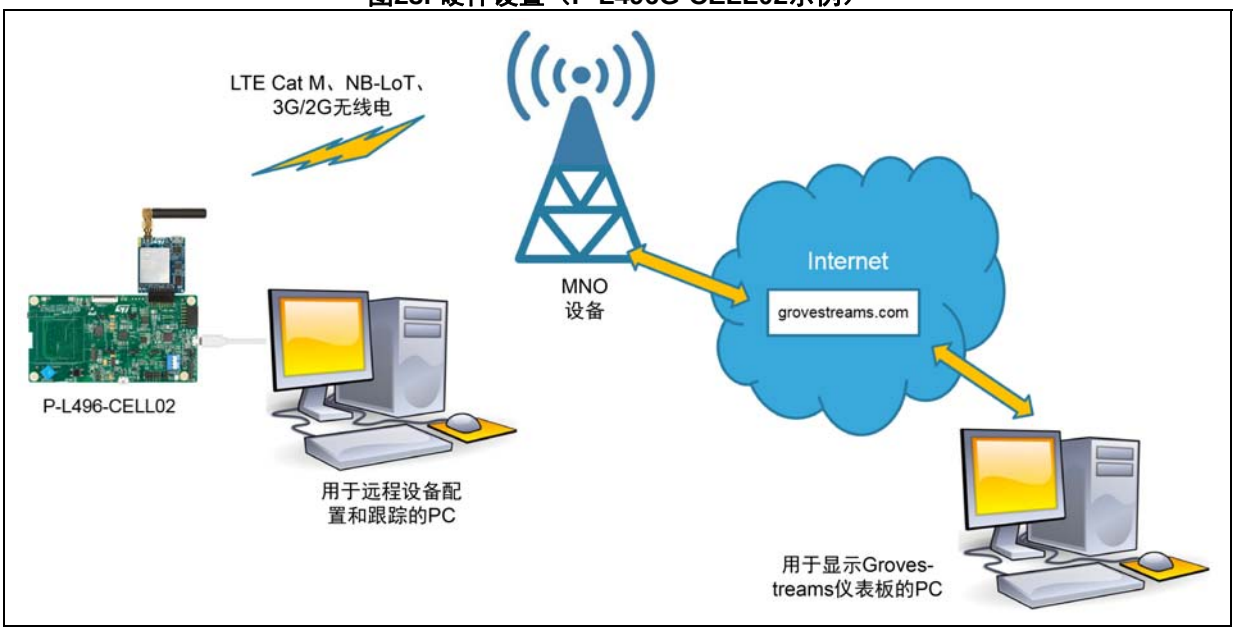
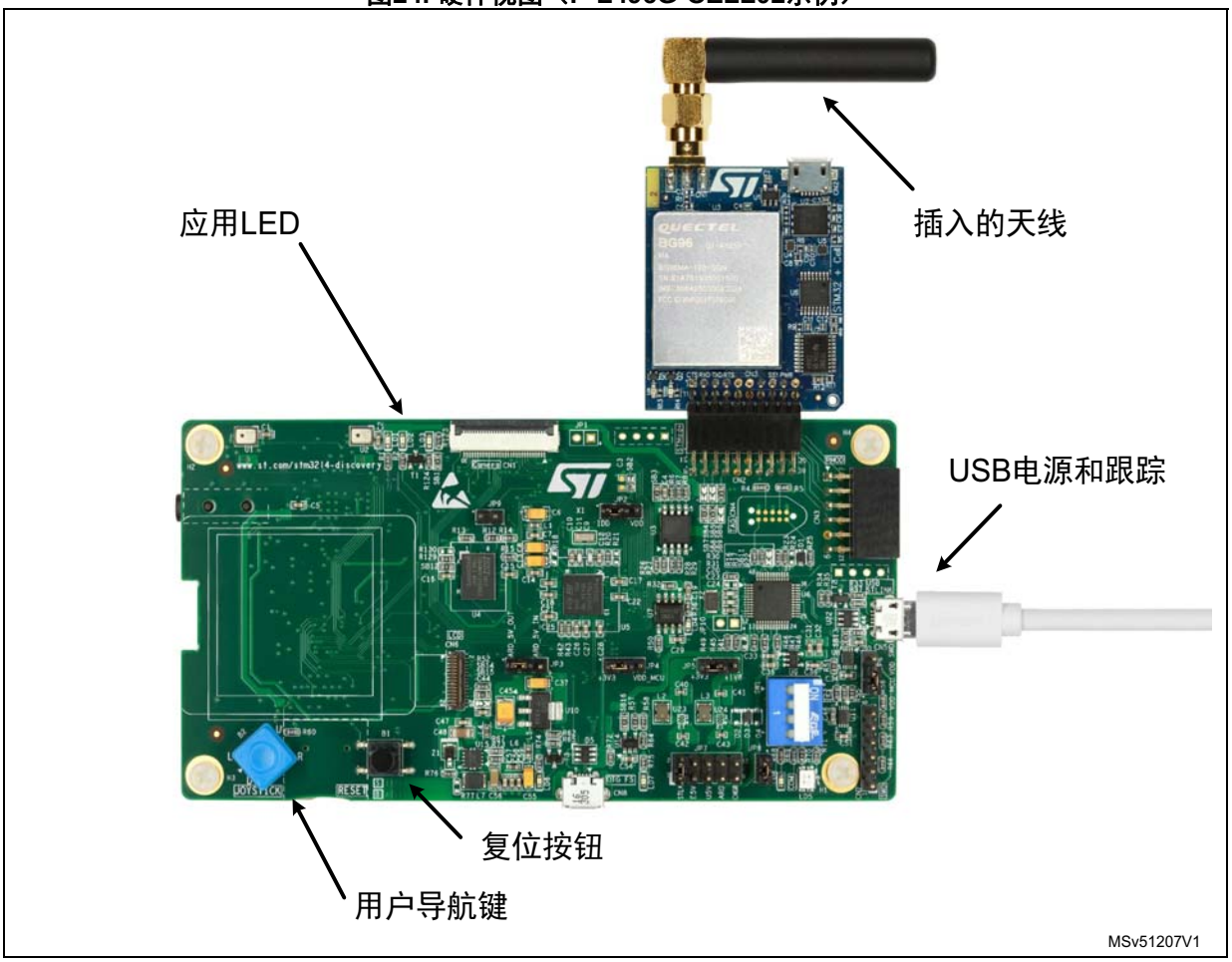


图24. 硬件视图（P-L496G-CELL02示例）



环境设置步骤总结：

- 创建Grovestreams帐户（参见*A.1：如何配置Grovestreams帐户第 56页*）
- 在Grovestreams帐户中创建一个Grovestreams组织（参见*A.1*）
- 获取组织中2个需要的API密钥（参见*A.1*）
- 用之前读取的用户API密钥更新提供的*.txt*文件
- 将调制解调器附加板及其天线连接到主机STM32板（参见*图 24*）
- 使用USB线将STM32板连接到PC（参见*图 24*）
- 通过将STM32 FW映像拖放到新出现的驱动器上来对STM32板进行编程
- 启动终端并设置参数（参见*第 7节：与主机板交互*）
- 重启电路板（按黑色按钮）
- 在启动菜单中选择项目“2”（有关完整说明，请参见*第 7节：与主机板交互*），选择要使用的SIM卡，并根据需要选择APN，设置Grovestreams参数，并保存到闪存中。APN名称由M(V)NO提供。

- 重启电路板（在终端中显示跟踪）
- 连接到Grovestreams帐户，选择仪表板，双击*传感器*，观察显示的数据，以及Grovestreams仪表板引起的LED切换。

Grovestreams有两个重要的txt文件：

- *GS_ Blueprint.txt*：在组织创建期间使用（按原样使用，无需修改）
- *GS_setup.txt*：用于配置通用FW，以访问Grovestreams帐户。必须通过将2个API密钥替换成组织的API密钥来更新此文件（参见*A.1：如何配置Grovestreams帐户第 56页*）

注： *“GS_setup.txt”文件不是必须的。这是一种将个人Grovestreams参数输入FW的一种用户友好方法，可避免通过使用“文件”>“发送文件…”并选择“GS_setup.txt”来单独输入每个参数。*

有关通过使用启动菜单（设置菜单 > Grovestreams）用文件“GS_setup.txt”输入Grovestreams参数的方法，请参见第 7节：与主机板交互。

有关通过使用启动菜单（设置菜单 > 蜂窝服务）选择UICC或SIM插槽及APN，请参见第 7节：与主机板交互。

7 与主机板交互

要与主机板交互，请使用串行控制台（USB虚拟COM端口）。对于Windows®操作系统，推荐使用Teraterm软件。

用于与主机板通信的串行端口设置如*图 25*中所示。通过*设置* > *串行端口*访问菜单。将*波特率*设为115200，以使Teraterm启动并运行。对于启动设置配置，必须应用10 ms的*发送延迟*。

图25. 与主机板交互的串行端口设置

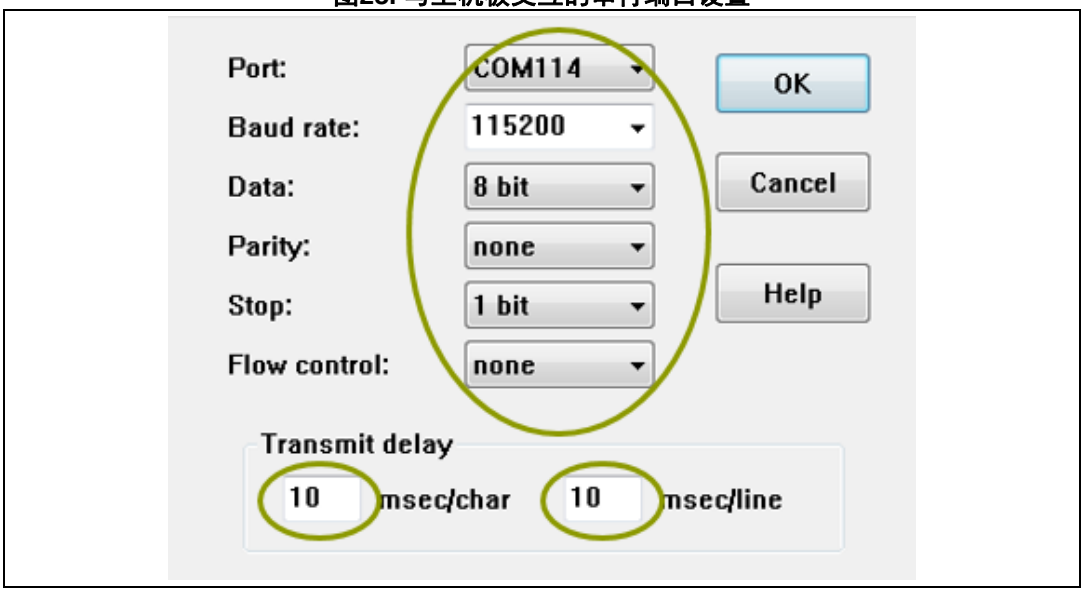
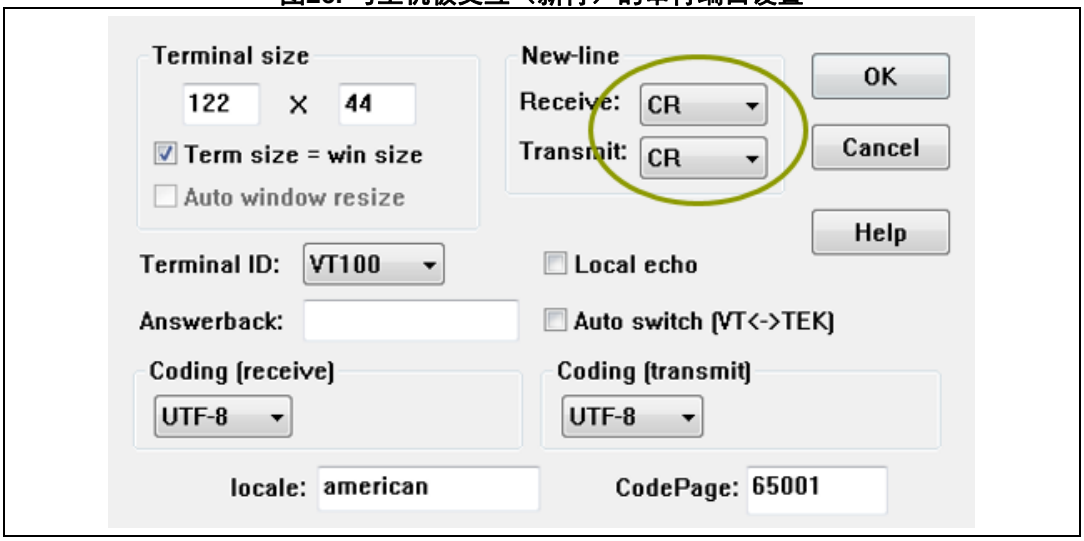


*图 26*说明了用于设置终端参数的菜单。可通过*设置* > *终端*来访问。将*接收*和*发送新行*参数设为CR。

图26. 与主机板交互（新行）的串行端口设置



在正确设置所有终端和串行端口参数后，按下电路板上的复位按钮（黑色）。

只要STM32 SW运行或向Grovestreams云发送数据，HTTP服务就有效。Grovestreams仪表板显示数据值，并打开/关闭设备LED。

可在启动时通过控制台设置参数。有关详细信息，请参见*第 7.2节：启动菜单*。

7.1 调试

通过连接到电路板并为其供电的PC虚拟COM端口查看调试跟踪。

可在*plf_sw_config.h*文件中自定义调试级别。*第 8.2.3节：激活和选择跟踪*和*第 8.4节：线程栈消耗监测*中提供了详细信息。

7.2 启动菜单

通过将配置文件中的USE_DEFAULT_SETUP变量设为0可以使能启动菜单，这是默认值*Projects\STM32L496G-Discovery\Common_Projects_Files\inc\plf_sw_config.h*.

在启动阶段，通过串行控制台显示一个菜单，以便选择固件使用：

0：*启动* 固件立即启动，而不会等待3秒

1：*设置菜单* 配置设置

2：*调制解调器上电* 仅启动调制解调器，而不启动固件应用程序

如果3秒后没有输入字符，则固件正常启动。

*表 3*说明了启动菜单。

表2. 启动菜单

```
==============================
      STM32 CELLULAR
      Version: xxx
      Date:    2018/xx/xx
==============================
Select the application to run:

0: Start
1: Setup Menu
2: Modem power on
```

7.2.1 “*启动*”选项

此选项可正常启动蜂窝服务，而不等待倒计时定时器。

7.2.2 “*调制解调器上电*”选项

调制解调器上电，而不启动蜂窝服务或应用程序。这对烧写调制解调器FW很有用。在上电后，如果将USB线连接到调制解调器板，则可以使用适当的PC SW更新调制解调器FW。

7.2.3 “*设置菜单*”选项

选择“*设置菜单*”选项可通过串行控制台显示设置菜单，如*表 3*中所示。

表3. 启动设置菜单

```
--------------------------------
   Setup Menu
--------------------------------
Select the component to config:
0: Quit
1: Date/Time (RTC)
2: Cellular Service
3: Grovestreams
8: Get list of config sources
9: Erase all feeprom(1) config
```

1. feeprom代表FEEPROM，它是主机MCU的嵌入式闪存。

0：退出

退出菜单并启动FW。

1：日期/时间（RTC）

用于设置系统日期和时间（GMT）。

根据Day, dd Month yyyy hh:mm:ss格式输入新的日期与时间。

示例：Mon, 11 Dec 2017 17:22:05

2：蜂窝服务

用于设置蜂窝服务参数：

- APN/CID关联
- SIM插槽选择
- NFMC特性在*[4]*中定义

有关蜂窝服务配置参数设置，请参见*第 7.2.4 节：蜂窝服务和Grovestreams配置子菜单*和*第 7.2.6 节：蜂窝服务配置参数*。

3: Grovestreams

设置HTTP客户端用于连接Grovestreams服务器的Grovestreams参数：

- Grovestreams URL
- API密钥 PUT和GET HTTP请求的设备标识符
- 传感器列表 用于从其获取值的传感器列表

有关Grovestreams配置参数设置，请参见*第 7.2.4 节：蜂窝服务和Grovestreams配置子菜单*和*第 7.2.7 节：Grovestreams配置参数*。

8：获取配置源列表

用于获取可用配置及其来源的列表（存储在FEEPROM中或默认值）。

*表 4*显示了选项*8：获取配置源列表*的示例，其中包括蜂窝服务组件的FEEPROM配置以及Grovestreams组件的默认配置。

表4. 配置源列表示例

```
---------------------------------
   List of config sources
---------------------------------
Cellular Service Config from FEEPROM
Grovestreams Config from DEFAULT
```

9：擦除所有feeprom配置

用于擦除存储在FEEPROM中的所有设置配置并恢复默认设置。

7.2.4 蜂窝服务和Grovestreams配置子菜单

*表 5*显示了为蜂窝服务或Grovestreams配置显示的配置子菜单。

表5. 配置子菜单

```
---------------------------------
  Setup Menu - Cellular Service
---------------------------------
 c : set config by console
 e : erase config in flash (restore default)
 l : list config
 q : quit
```

c：通过控制台设置配置

用于修改配置并将其存储在FEEPROM中。

e：擦除闪存中的配置

擦除存储在FEEPROM中的配置并恢复默认设置。

l：列出配置

列出当前配置。

q：退出菜单

返回上一级菜单。

7.2.5 通过控制台选项设置配置

当通过配置子菜单中的选项*c*选择*通过控制台设置配置*选项时，必须在控制台上输入参数值。

有三种方法可用于输入参数值：

- 通过键盘手动输入配置
- 从配置文件复制并粘贴配置
- 使用Teraterm*发送*菜单将配置文件发送到控制台

除*版本*以外，对于每个参数，将显示参数的当前值。如果未输入任何值（按下返回键），则保留当前值。

*版本*是第一个配置字段。它允许在使用配置文件时检查配置版本。如果版本不匹配，则中止配置。

- 如果手动输入配置，则必须键入第一个配置行上显示的版本
- 如果从文本文件输入配置，请检查文件的配置（第一行）是否与控制台显示的配置版本匹配

示例：

- 版本(2): 2

注意： 不同固件版本的配置格式可能有所不同。在这种情况下，首次启动时会擦除存储在FEEPROM中的配置，并恢复默认配置。

7.2.6 蜂窝服务配置参数

版本（*n*）

配置格式的版本（参见*第 7.2.5 节：通过控制台选项设置配置*）。

pdn配置模式

用于选择APN和CID定义模式。

- 0：仅使用CID值并将该值传送到调制解调器。
  仅当先前已将APN/CID关联存储在调制解调器中时，才能使用此选项。
- 1：关联定义的APN和CID（参见*APN*和*CID*），并将其发送至调制解调器。

*pdn配置模式*的默认值为1。

注意： 如果调制解调器无法永久存储配置，则必须让*pdn配置模式*的值保持为1。这是UG96调制解调器的情况。

如果调制解调器具有永久存储配置的能力，则可以使用*pdn配置模式*设置值1对调制解调器进行一次配置。随后可将该参数设为0。

APN

与CID关联的APN。

仅在将*pdn配置模式*参数设为1时才考虑此参数（参见*pdn配置模式*）。

默认情况下，APN参数为空字符串。如果输入非空字符串，则再次设置空字符串的唯一方法是擦除蜂窝服务闪存配置（参见*e：擦除闪存中的配置*）。

*APN*的默认值为空字符串。

CID

要使用的CID值。

- 如果将*pdn配置模式*参数设为1，则CID与APN相关联
- 如果将*pdn配置模式*设为0，则不使用APN，并且必须先将关联的APN存储在调制解调器中。

有关CID/APN关联的详细信息，请参见*pdn配置模式*。

*CID*的默认值为1。

Sim插槽。

SIM插槽选择。

- 0：插槽
- 1：嵌入式SIM插槽
- 2：主机SIM插槽（未实现）

*Sim插槽*的默认值为0。

NFMC激活

用于启用和禁用NFMC功能。

- 0：已禁用NFMC特性
- 1：已启用NFMC特性。使用基本临时参数。

*NFMC激活*的默认值为0。

基本临时参数的默认值为：

- NFMC值1：60000 ms
- NFMC值2：120000 ms
- NFMC值3：240000 ms
- NFMC值4：480000 ms
- NFMC值5：960000 ms
- NFMC值6：192000 ms
- NFMC值7：3840000 ms

有关EMnify蜂窝配置示例，请参见 *Utilities\PC_Software\Tools\Cellular_Service\emnify_cellular_config.txt*。

7.2.7 Grovestreams配置参数

版本（*n*）

配置格式的版本（参见*第 7.2.5节：通过控制台选项设置配置*）。

要联系的主机

主机服务器和端口的URL或IP地址。

示例：

- 输入要联系的主机的IP（xxx.xxx.xxx.xxx:xxxxx）
- 输入要联系的主机的URL（www.url.com port）
  例如：www.grovestreams.com 80

PUT API Key

HTTP PUT请求的API密钥。用于将传感器值发送到Grovestreams站点。通过Grovestreams站点获取API密钥（参见*A.1：如何配置Grovestreams帐户第 56页*）。

GET API Key

HTTP GET请求的API密钥。用于获取在Grovestreams站点上设置的值（LED状态）。通过Grovestreams站点获取API密钥（参见*A.1：如何配置Grovestreams帐户第 56页*）。

组件ID

用于Grovestreams配置的设备组件ID。

*组件ID*的默认值为Sensors_DEF。

“PUT请求”周期

PUT请求的周期以秒表示。如果不需要PUT请求，请将此参数设置为0。

*“PUT请求”周期*的默认值为25，最小值为10。

要发送的传感器值列表

这是传感器类型与Grovestreams中配置的相关标识符之间的关联列表。

示例：在设备上，电池电量类型为1。在Grovestreams配置中，如果将电池电量标识符设为batlevel，则关联参数为：

- 传感器类型：1
- 通道ID：batlevel

输入0作为*作为传感器类型*，以退出列表。

传感器类型列表：

- 0：结束PUT请求配置
- 1：电池电量
- 2：调制解调器信号电平
- 3：HRM心率（未实现）
- 4：计步器（未实现）
- 5：湿度（插入Sensor Shield时可用）
- 6：湿度（插入Sensor Shield时可用）
- 7：温度（插入Sensor Shield时可用）
- 8：压力（插入Sensor Shield时可用）

默认关联值为1/batlevel和2/siglevel。

“GET请求”周期

GET请求的周期以秒表示。如果不需要GET请求，请将此参数设置为0。

*“GET请求”周期*的默认值为2。

要获取的值列表

这是从Grovestreams获取的值类型与Grovestreams中配置的相关标识符之间的关联列表。

示例：在设备上，LED状态类型为1。在Grovestreams配置中，如果将电池电量标识符设为ledlight，则关联参数为：

- 传感器类型：1
- 通道ID：ledlight

输入0作为*作为传感器类型*，以退出列表。

传感器类型列表：

- 0：结束GET请求配置
- 1：LED状态

默认关联值为1/ledlight。

7.2.8 启动菜单和配置完成示例

==========

STM32 CELLULAR

Version: xxx

Date: 2018/xx/xx

==========

Select the application to run:

0: Start

1: Setup Menu

2: Modem power on

Date: Mon 00-1-1 - 02:00:42

Setup Menu

Select the component to config:

0: Quit

1：日期/时间（RTC）

2: Cellular Service

3: Grovestreams

8: Get list of config sources

9: Erase all feeprom config

Setup Menu - Grovestreams

---

c : set config by console

e : erase config in flash (restore default)

l : list config

q : quit

Grovestreams Config from UART

Enter Grovestreams GET Key (e02848d7-0e7d-3fc4-9bd6-22f62d3c4xxx): cc706cd8-ebab-3029-8ada-722d879a2xxx

Enter Grovestreams PUT Key (b92a7d9b-1bb3-38cf-8ea2-c1e0ff0dfxxx): e02848d7-0e7d-3fc4-9bd6-22f62d3c4xxx

Enter Component ID (comp1): comp1

PUT request

Enter PUT request period (25): 25

Config a sensor

Select sensor type

0 : quit

Enter Selection (0 to quit) (1): 1

Enter channel ID of sensorEnter Channel ID (batlevel): batlevel

Config a sensor

Select sensor type

0 : quit

Enter Selection (0 to quit) (2): 2

Enter channel ID of sensorEnter Channel ID (siglevel): siglevel

Config a sensor

Select sensor type

0 : quit

Enter Selection (0 to quit) (5): 5

Enter channel ID of sensorEnter Channel ID (hum): humidity

Config a sensor

Select sensor type

0 : quit

Enter Selection (0 to quit) (6): 6

Enter channel ID of sensorEnter Channel ID (temp): temperature

Config a sensor

Select sensor type

0 : quit

Enter Selection (0 to quit) (7): 7

Enter channel ID of sensorEnter Channel ID (press): pressure

Config a sensor

Select sensor type

 0 : quit

Enter Selection (0 to quit) (0): 0

Enter GET request period (2): 2

Config a sensor

Select sensor type

 0 : quit

Enter Selection (0 to quit) (1): 1

Enter channel ID of sensorEnter Channel ID (ledlight): ledlight

Config a sensor

Select sensor type

 0 : quit

Enter Selection (0 to quit) (0): 0

save config in feeprom ?(y/n) :y

本示例还包括所有可能的选定项目（5 PUT和1 GET）。它可作为模板文本文件复制，也可以定制到Teraterm安装文件夹中（默认为*C:\Program Files (x86)\teraterm*\），默认情况下在使用Teraterm的*发送文件*选项时提供。

8 如何定制软件

本节介绍了三种可能适用于X-CUBE-CELLULAR的软件定制级别：用户定制、高级用户定制和开发人员定制。本章也介绍了如何监测线程栈消耗。

8.1 第一个定制级别：用户定制

第一个定制级别为启动时的设置配置（参见*第 7节：与主机板交互第 40页*）。

在该级别中，不修改固件。无需定制引起的编译。

8.2 第二个定制级别：高级用户定制

在该级别中，可以修改固件配置。可添加或删除特定功能，并可以修改固件配置参数，如章节*8.2.1*、*8.2.2*和*8.2.3*中所示。

需要定制引起的重新编译。

8.2.1 向固件添加/删除应用程序

Projects\STM32L496G-Discovery\Common_Projects_Files\inc\plf_features.h 配置文件允许选择将包含在固件中的应用程序。

*表 6*提供了可根据所需应用程序的函数定义或取消定义的编译变量。

表6. 固件中的应用程序编译变量

编译变量(1)	说明
#define USE_HTTP_CLIENT	包括HTTP客户端（Grovestreams）应用程序。
#define USE_PING_CLIENT	包括ping实用程序。ping应用程序使用了COM PING API功能。 如果定义了USE_PING_CLIENT，也必须定义USE_COM_PING（参见此表中的相应条目）。
#define USE_DC_MEMS	包括传感器管理 *Projects\Common\tests_utilities\dc_mems.c*. 注：此选项只能在32L496GDISCOVERY上设置。
#define USE_SIMU_MEMS	包括传感器仿真（无可用的物理传感器） *Projects\Common\tests_utilities\dc_mems.c* 此选项也可用于定义的USE_DC_MEMS。在未插入Sensor Shield时有用。
#define USE_DEFAULT_SETUP	定义是否使用启动设置菜单： – 0：使用启动设置菜单 – 1：不使用启动设置菜单。在该情况下，设置默认参数。使用设置菜单在每个组件的*<component>config.h*文件中定义这些参数。 *Projects\Applications\Cellular\radio_service\cellular\inc\cellular_service_config.h* *Projects\Common\applications\HTTP\inc\httpclient_config.h*

表6. 固件中的应用程序编译变量（续）

编译变量(1)	说明
#define USE_DC_EMUL	包括数据缓存测试的传感器仿真 *Projects\Common\tests_utilities\dc_emul.c*
#define USE_DC_TEST	包括数据缓存测试 *Projects\Common\tests_utilities\dc_test.c*
#define USE_COM_PING	包括模块COM中的ping功能。 由于Ping功能只使用少量的内存，因此该定义提供如果在平台上不使用Ping则不包含Ping功能的可能性。 如果定义了USE_PING_CLIENT，则必须定义USE_COM_PING。
#define COM_SOCKETS_ERRNO_COMPAT	如果激活，则在将USE_SOCKETS_TYPE设为USE_SOCKETS_MODEM时，带COM_SO_ERROR参数的com_getsockopt返回与*errno.h*兼容的值（有关转换，请参见*com_sockets_err_compat.c*文件）

1. 所有定义都是独立的：可以一起选择多个应用程序。

8.2.2 MCU侧或调制解调器侧的IP栈

IP栈在MCU侧或调制解调器侧运行。默认配置为：调制解调器侧

*Projects\STM32L496G-Discovery\Common_Projects_Files\inc\plf_feature.h*配置文件包括允许定义所用IP栈的位置。

*表 7*提供了定义所用IP栈的编译变量。

表7. IP栈选择编译变量

编译变量	说明
#define USE_SOCKETS_TYPE	定义使用的IP栈： – USE_SOCKETS_LWIP： MCU侧的IP栈（LwIP栈） – USE_SOCKETS_MODEM： 调制解调器端的IP栈(1)

1. 如果使用的IP堆栈位于调制解调器端，则仅支持TCP IPv4客户端。

8.2.3 激活和选择跟踪

*Projects\STM32L496G-Discovery\Common_Projects_Files\inc\plf_sw_config.h*配置文件包括用于选择跟踪行为的参数：

*表 8*提供了定义跟踪配置的编译变量。

表8. 跟踪编译变量

编译变量	说明
#define TRACE_IF_TRACES_ITM	跟踪ITM上的输出（默认启动）
#define TRACE_IF_TRACES_UART	跟踪UART上的输出（默认启动）
#define USE_TRACE_<XXX>	激活/停用组件<XXX>跟踪

8.3 第三个定制级别：开发人员定制

8.3.1 BOOT

启动是设备初始化的第一部分。它包含在*main.c*文件中（主函数）。此部分涉及HAL所进行的HW初始化。此文件由STM32CubeMX生成，并提供.*ioc*文件。仅在使用新HW时或主机板用户配置外设（如GPIO、I^2C）时更新。

8.3.2 软件组件初始化

SW组件（应用和中间件）在*freeRTOS.c*文件中初始化。每个组件均包含用于初始化数据结构的静态初始化（<Component>_Init）以及用于启动组件线程的实时初始化（<Component>_Start）。两者都以正确的顺序从StartDefaultTask函数中调用。

8.3.3 软件定制

固件配置参数包含在以下文件中：

- *FreeRTOSConfig.h*： 包含FreeRTOS™参数
- *lwipopts.h*： 包含LwIP参数
- *plf_hw_config.h*： 包含HW参数（如UART配置、使用的GPIO等）。通常需要进行更改，以使软件适应新的电路板。
- *plf_sw_config.h*： 包含SW参数（如任务优先级、跟踪激活等）
- *plf_stack_size.h*： 包括线程栈大小。本文件中包含的栈大小用于计算FreeRTOS™堆大小（包含在*FreeRTOSConfig.h*文件中）。
- *plf_features.h*： 包含选定应用程序

8.3.4 新HW配置的固件适配

要为电路板或HW配置适配固件，请按以下步骤操作：

1. 创建基于新主板的STM32CubeMX项目
2. 按对32L496GDISCOVERY板的配置方式配置HW IP
3. 通过STM32CubeMX生成软件配置文件
4. 更新*plf_hw_config.h*文件，以匹配配置文件中生成的GPIO和HW处理程序名称

8.3.5 添加新组件

要添加新组件（应用或中间件），请遵循以下步骤：

1. 创建初始化函数<Component>_Init，以初始化应用程序的数据结构
2. 创建启动函数<Component>_Start，以启动组件线程
3. 在StartDefaultTask函数（*freertos.c*文件中）中添加<Component>_Init和<Component>_Start调用
4. 在*plf_sw_config.h*文件中添加栈优先级常量，并在*plf_stack_size.h*文件中添加栈大小。也可以添加其他方便的配置编译变量。
5. 实现组件核心。每个组件至少拥有一个线程。

   使用数据缓存完成与其他组件和整个系统的交互。数据缓存包含要在组件之间共享的所有系统数据（传感器值、网络状态等）。当组件更新数据缓存中的数据时，将调用所有已订阅的回调。

   通常，组件会订阅数据缓存回调。应用线程的核心等待数据缓存事件并对其进行处理。

*httpclient.c*文件可用作创建新应用的示例。

8.4 线程栈消耗监测

栈分析模块可以监控线程栈消耗。

每次创建线程时，必须将其注册添加到栈分析，以提供其栈大小分配（已对项目中所声明的所有线程执行该操作）。

*表 9*显示，对于任何新线程创建，将调用添加到stackAnalysis_addThreadStackSizeByHandle()服务中。

表9. 新线程注册示例

```
osThreadDef(defaultTask, StartDefaultTask, CTRL_THREAD_PRIO, 0,
            CTRL_THREAD_STACK_SIZE);
defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
#if (STACK_ANALYSIS_TRACE == 1)
     stackAnalysis_addThreadStackSizeByHandle(defaultTaskHandle,
CTRL_THREAD_STACK_SIZE);
#endif /* STACK_ANALYSIS_TRACE */
```

如果需要，可增加*plf_sw_config.h*文件中的项目线程数，如*表 10*中所示。

表10. 项目线程数设置示例

```
/* Number of threads in the Project */
#define THREAD_NUMBER 20 /* Thread defined below + Timer + Idle */
```

为监测软件执行期间的线程栈消耗，在初始化结束时将*表 11*中显示的行添加到StartDefaultTask。

表11. 线程栈消耗监测代码

```
    用于(;;)
    {
#if (STACK_ANALYSIS_TRACE == 1)
        stackAnalysis_trace(true);
        /* print stack analysis status every 5 sec*/
        osDelay(5000);
#else
        osDelay(1000);
#endif
    }
```

默认情况下不激活监测。要激活监测（所有线程栈大小演进的printf），在*plf_sw_config.h*文件中将

#define STACK_ANALYSIS_TRACE 0

更改为

#define STACK_ANALYSIS_TRACE 1

监测默认配置会导致每5隔秒打印一次所有线程栈大小的最大占用。

有关涉及配置监测的更多选项，请参见*stack_analysis.h.*

附录A 辅助资料

A.1 如何配置Grovestreams帐户

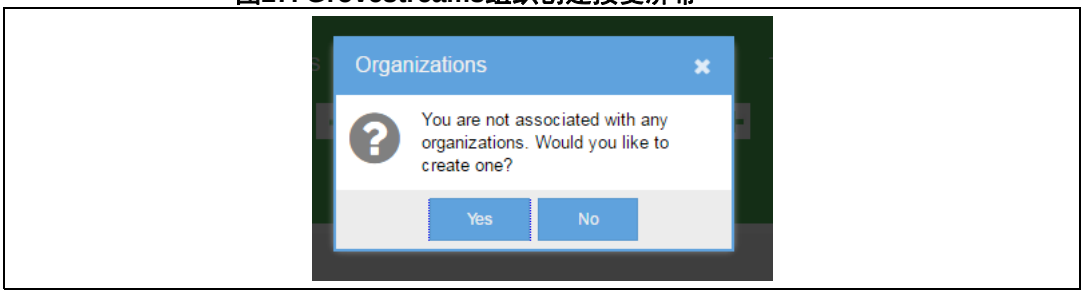
不同的步骤为：

1. 如果需要，创建一个电子邮件帐户（创建Grovestreams帐户需要有效的电子邮件帐户）
2. 创建Grovestreams帐户（免费注册）
3. 设置Grovestreams帐户（使用蓝图创建组织）
4. 获取所需的API密钥（用于STM32 MCU FW）

*图 27*到*图 32*的一系列数据说明了Grovestreams帐户的创建和配置。

*图 27*显示了创建Grovestreams组织的登录屏幕。

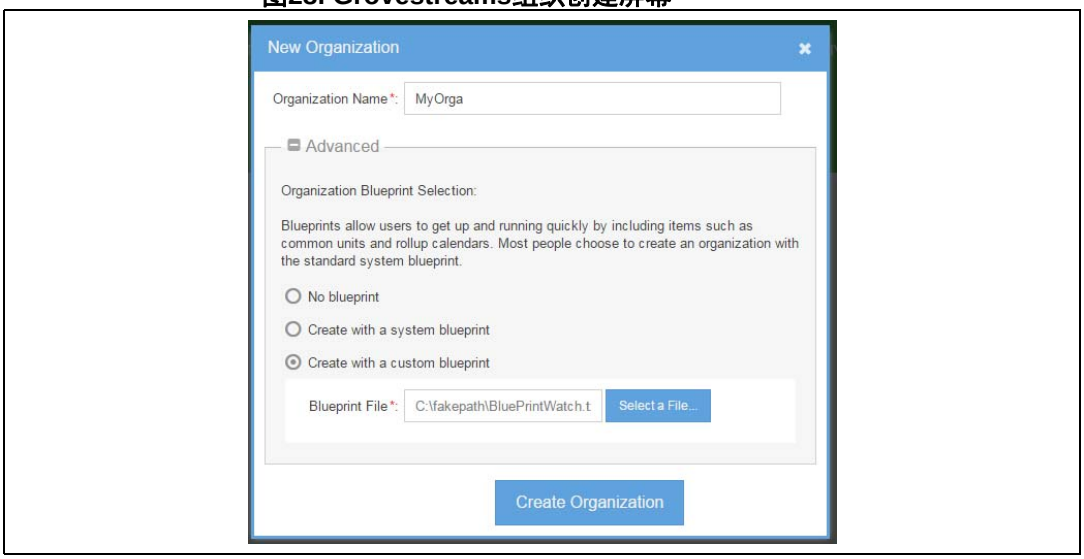
图27. Grovestreams组织创建接受屏幕



用于创建组织的蓝图（模板）如*图 28*中所示：

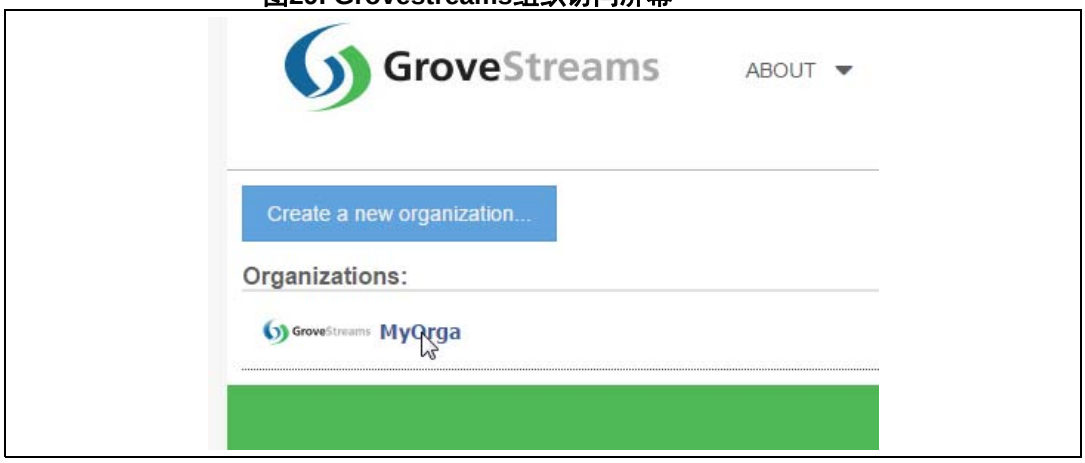
- 在“*组织名称*”字段中输入组织名称
- 勾选*使用自定义蓝图创建*选项
- 选择*蓝图文件*
- 使用*创建组织*进行验证

图28. Grovestreams组织创建屏幕



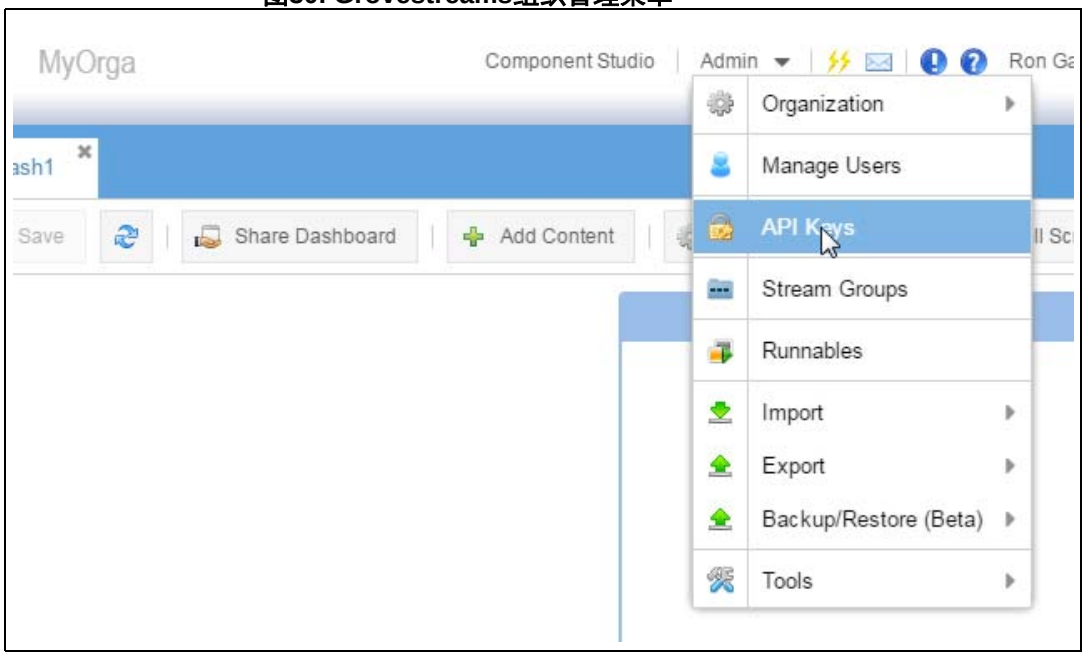
访问*图 29*中所示的组织。

图29. Grovestreams组织访问屏幕



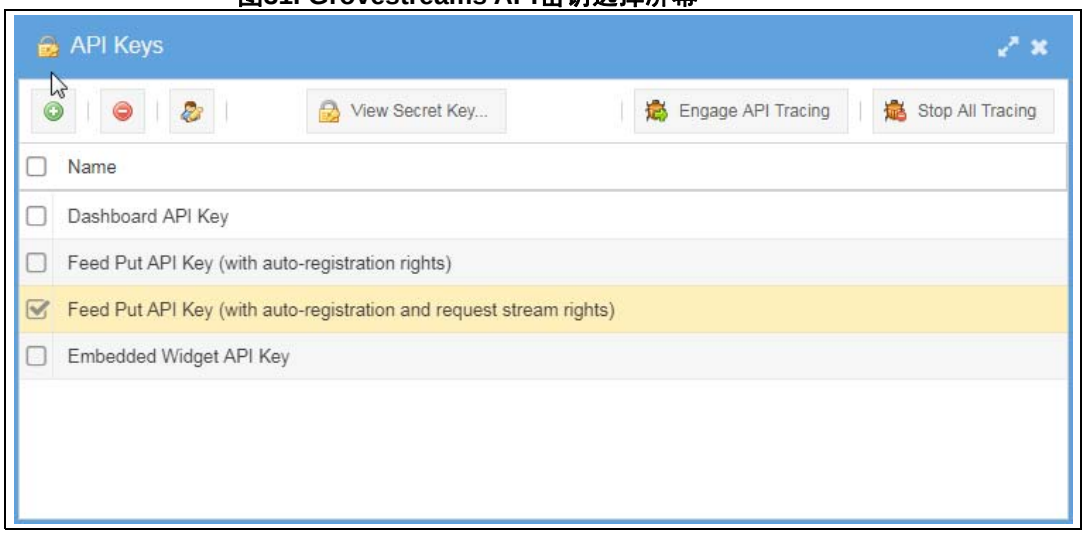
通过选择*Admin* > *API密钥*准备复制API密钥，如*图 30*中所示。

图30. Grovestreams组织管理菜单



选择*Feed Put API Key*，如*图 31*中所示。

图31. Grovestreams API密钥选择屏幕



选择*View Secret Key*以显示密钥，如*图 32*中所示。

图32. Grovestreams API密钥显示屏幕

可根据需要复制和粘贴API密钥。对仪表板API密钥执行类似的操作，在选择*Dashboard API Key*之前取消选择*Feed Put API Key*。

A.2 如何激活焊接的SIM卡

C2C套件提供由EMnify MVNO预先配置的UICC芯片。这允许全球2G和3G漫游。这不是eUICC，而是嵌入式SIM。这是在工厂过程中在芯片中配置的经典SIM配置文件。这不是允许远程配置文件更新的eUICC。

EMnify不支持LTE Cat M1和NB1技术。如果在BG96板上激活UICC，则只能进行2G回落。

要在Cat M1或NB1模式下使用BG96调制解调器，用户必须使用microSIM塑料卡。

UG96附加板没有限制。

在UICC芯片上激活EMnify的步骤如下：

1. 在连接终端的情况下启动电路板，以便可以读取文本
2. 从显示的文本中获取凭证
3. 通过PC连接到https://stm32-c2c.com
4. 使用凭证注册电路板
5. 在显示的电路板中选择EMnify项
6. 执行激活步骤
7. EMnify配置文件会立即激活（双击SIM卡以显示SIM状态）

A.3 常见问题

问：在哪里可以找到用于创建Grovestreams组织的蓝图文件？

答：*/Utilities/PC_Software/Grovestreams*中的*GS_Blueprint.txt*文件。

问：在哪里可以找到用于更新Grovestreams参数的文本文件？

答：*/Utilities/PC_Software/Grovestreams*中的*GS_Setup.txt*文件。

问：如何知道蜂窝应用已启动并运行？

答：使用Teraterm上的跟踪，或连接到Grovestreams帐户并检查数据是否已更新。

问：Teraterm中没有显示任何内容。

答：确保正确设置Teraterm选项（如*波特率*）。

问：设置期间Teraterm中出现错误回显。

答：确保正确设置Teraterm选项（如*新行*和*发送延迟*）。

问：启动但出现网络问题。

答：检查是否选择正确的SIM（塑料SIM或焊接的UICC；默认情况下为塑料SIM）。

问：如何检查UICC上的EMnify配置文件是否已正确激活？

答：连接到EMnify帐户并检查状态。

问：EMnify已正确激活，但不工作。

答：如果已选择正确的费率，请检查EMnify帐户。

问：日志表明Grovestreams已正常启动并运行，但没有任何变化。

答：检查是否使用正确的API密钥（与用于演示的组织相关的API密钥）。

问：由于初始映像被覆盖，因此无法读取凭证。

答：浏览至www.stm32-c2c.com并下载*出厂恢复固件*。

问：在使用开箱即用的FW时，控制台上无任何信息。

答：在Teraterm中设置正确的参数（将*波特率*设为9600，以获得凭证），并确保调制解调器未通过STMod+连接器倒置连接到主机板。

A.4 X-CUBE-CELLULARAPI描述

本节介绍了上层（应用程序）所使用的API。

此API具有2个模块，即COM和数据缓存。

A.4.1 COM API

本节中的一些软件是从*Application\net\com\inc\com_sockets.h文件中提取。*

套接字管理

表12. COM API管理 - 创建套接字句柄

```
/**
  * @brief  Socket handle creation
  * @note   Create a communication endpoint called socket
  * @param  family   - address family
  * @param  type     - connection type
  * @param  protocol - protocol type
  * @retval int32_t  - socket handle or error value
  */
int32_t com_socket(int32_t family, int32_t type, int32_t protocol);
```

表13. COM API管理 - 设置套接字选项

```
/**
  * @brief  Socket option set
  * @note   Set option for the socket
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  level     - level at which the option is defined
  * @param  optname   - option name for which the value is to be set
  * @param  optval    - pointer to the buffer containing the option value
  * @param  optlen    - size of the buffer containing the option value
  * @retval int32_t   - ok or error value
  */
int32_t com_setsockopt(int32_t sock, int32_t level, int32_t optname,
                       const void *optval, int32_t optlen);
```

表14. COM API管理 - 获取套接字选项

```
/**
  * @brief  Socket option get
  * @note   Get option for a socket
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  level     - level at which option is defined
  * @param  optname   - option name for which the value is requested
  * @param  optval    - pointer to the buffer that will contain the option value
  * @param  optlen    - size of the buffer that will contain the option value
  * @retval int32_t   - ok or error value
  */
int32_t com_getsockopt(int32_t sock, int32_t level, int32_t optname,
                       void *optval, int32_t *optlen);
```

表15. COM API管理 - 套接字绑定

```
/**
  * @brief  Socket bind
  * @note   Assign a local address and port to a socket
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  addr      - local IP address and port
  * @param  addrlen   - addr length
  * @retval int32_t   - ok or error value
  */
int32_t com_bind(int32_t sock,
                 const com_sockaddr_t *addr, int32_t addrlen);
```

表16. COM API管理 - 关闭套接字

```
/**
  * @brief  Socket close
  * @note   Close a socket and release socket handle
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @retval int32_t   - ok or error value
  */
int32_t com_closesocket(int32_t sock);
```

客户端功能

表17. COM API客户端 - 套接字连接

```
/**
  * @brief  Socket connect
  * @note   Connect socket to a remote host
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  addr      - remote IP address and port
  * @param  addrlen   - addr length
  * @retval int32_t   - ok or error value
  */
int32_t com_connect(int32_t sock,
                    const com_sockaddr_t *addr, int32_t addrlen);
```

表18. COM API客户端 - 套接字发送数据

```
/**
  * @brief  Socket send data
  * @note   Send data on already connected socket
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  buf       - pointer to application data buffer to send
  * @param  len       - size of the data to send (in bytes)
  * @param  flags     - options
  * @retval int32_t   - number of bytes sent or error value
  */
int32_t com_send(int32_t sock,
                 const com_char_t *buf, int32_t len, int32_t flags);
```

表19. COM API客户端 - 套接字接收数据

```
/**
  * @brief  Socket receive data
  * @note   Receive data on already connected socket
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  buf       - pointer to application data buffer to store the data to
  * @param  len       - size of application data buffer (in bytes)
  * @param  flags     - options
  * @retval int32_t   - number of bytes received or error value
  */
int32_t com_recv(int32_t sock,
                 com_char_t *buf, int32_t len, int32_t flags);
```

服务器功能

表20. COM API服务器 - 套接字监听

```
/**
  * @brief  Socket listen
  * @note   Set socket in listening mode
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  backlog   - number of connection requests that can be queued
  * @retval int32_t   - ok or error value
  */
int32_t com_listen(int32_t sock, int32_t backlog);
```

表21. COM API服务器 - 套接字接受

```
/**
  * @brief  Socket accept
  * @note   Accept a connect request for a listening socket
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  addr      - IP address and port number of the accepted connection
  * @param  len       - addr length
  * @retval int32_t   - ok or error value
  */
int32_t com_accept(int32_t sock,com_sockaddr_t *addr, int32_t *addrlen);
```

表22. COM API服务器 - 套接字发送数据

```
/**
  * @brief  Socket send to data
  * @note   Send data to a remote host
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  buf       - pointer to application data buffer to send
  * @param  len       - length of the data to send (in bytes)
  * @param  flags     - options
  * @param  addr      - remote IP address and port number
  * @param  len       - addr length
  * @retval int32_t   - number of bytes sent or error value
  */
int32_t com_sendto(int32_t sock,
                   const com_char_t *buf, int32_t len, int32_t flags,
                   const com_sockaddr_t *to, int32_t tolen);
```

表23. COM API服务器 - 套接字接收数据

```
/**
  * @brief  Socket receive from data
  * @note   Receive data from a remote host
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  buf       - pointer to application data buffer to store the data to
  * @param  len       - size of application data buffer (in bytes)
  * @param  flags     - options
  * @param  addr      - remote IP address and port number
  * @param  len       - addr length
  * @retval int32_t   - number of bytes received or error value
  */
int32_t com_recvfrom(int32_t sock,
                     com_char_t *buf, int32_t len, int32_t flags,
                     com_sockaddr_t *from, int32_t *fromlen);
```

其他功能

表24. COM API的其他功能 - 组件初始化

```
/**
  * @brief  Component initialization
  * @note   must be called only one time and
  *         before using any other functions of com_*
  * @参数 无
  * @retval bool      - true/false init ok/nok
  */
com_bool_t com_init(void);
```

表25. COM API的其他功能 - 组件启动

```
/**
  * @brief  Component start
  * @note   must be called only one time but
  *         after com_init and dc_start
  *         and before using any other functions of com_*
  * @参数 无
  * @返回值：无
  */
void com_start(void);
```

表26. COM API的其他功能 - 通过主机名称获取主机IP

```
/**
  * @brief  Get host IP from host name
  * @note   Retrieve host IP address from host name
  * @param  name      - host name
  * @param  addr      - host IP corresponding to host name
  * @retval int32_t   - ok or error value
  */
int32_t com_gethostbyname(const com_char_t *name,
                          com_sockaddr_t *addr);
```

表27. COM API的其他功能 - 获取对端名称

```
/**
  * @brief  Get peer name
  * @note   Retrieve IP address and port number
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  name      - IP address and port number of the peer
  * @param  namelen   - name length
  * @retval int32_t   - ok or error value
  */
int32_t com_getpeername(int32_t sock,
                        com_sockaddr_t *name, int32_t *namelen);
```

表28. COM API的其他功能 - 获取套接字名称

```
/**
  * @brief  Get sock name
  * @note   Retrieve local IP address and port number
  * @param  sock      - socket handle obtained with com_socket
  * @note   socket handle on which operation has to be done
  * @param  name      - IP address and port number
  * @param  namelen   - name length
  * @retval int32_t   - ok or error value
  */
int32_t com_getsockname(int32_t sock,
                        com_sockaddr_t *name, int32_t *namelen);
```

表29. COM API的其他功能 - Ping API

```
#if (USE_COM_PING == 1)
/**
  * @brief  Ping handle creation
  * @note   Create a ping session
  * @参数 无
  * @retval int32_t  - ping handle or error value
  */
int32_t com_ping(void);

/**
  * @brief  Ping process request
  * @note   Create a ping session
  * @param  ping     - ping handle obtained with com_ping
  * @note   ping handle on which operation has to be done
  * @param  addr     - remote IP address and port
  * @param  addrlen  - addr length
  * @param  timeout  - timeout for ping response (in sec)
  * @param  rsp      - ping response
  * @retval int32_t  - ok or error value
  */
int32_t com_ping_process(int32_t ping,
                         const com_sockaddr_t *addr, int32_t addrlen,
                         uint8_t timeout, com_ping_rsp_t *rsp);

/**
  * @brief  Ping close
  * @note   Close a ping session and release ping handle
  * @param  ping      - ping handle obtained with com_socket
  * @note   ping handle on which operation has to be done
  * @retval int32_t   - ok or error value
  */
int32_t com_closeping(int32_t ping);
#endif /* USE_COM_PING == 1 */
;
```

A.4.2 数据缓存API

数据缓存API分为几个文件，并涵盖了事件ID、数据ID和服务。

事件ID

表30. *dc_control.h*中的数据缓存API（事件ID）

```
extern dc_com_res_id_t    DC_COM_BUTTON_UP    ;
extern dc_com_res_id_t    DC_COM_BUTTON_DN    ;
extern dc_com_res_id_t    DC_COM_BUTTON_RIGHT ;
extern dc_com_res_id_t    DC_COM_BUTTON_LEFT  ;
extern dc_com_res_id_t    DC_COM_BUTTON_SEL   ;
```

蜂窝数据ID

表31. *dc_cellular.h*中的数据缓存API（蜂窝数据ID）

```
extern dc_com_res_id_t    DC_COM_CELLULAR      ;
extern dc_com_res_id_t    DC_COM_PPP_CLIENT    ;
extern dc_com_res_id_t    DC_COM_CELLULAR_DATA ;
extern dc_com_res_id_t    DC_COM_RADIO_LTE     ;
extern dc_com_res_id_t    DC_COM_NIFMAN        ;
extern dc_com_res_id_t    DC_COM_NFMC_TEMPO    ;
extern dc_com_res_id_t    DC_COM_SIM_INFO      ;
```

传感器数据ID

表32. *dc_mems.h*中的数据缓存API（传感器数据ID）

```
extern dc_com_res_id_t  DC_COM_PRESSURE      ;
extern dc_com_res_id_t  DC_COM_HUMIDITY      ;
extern dc_com_res_id_t  DC_COM_TEMPERATURE   ;
extern dc_com_res_id_t  DC_COM_ACCELEROMETER ;
extern dc_com_res_id_t  DC_COM_GYROSCOPE     ;
extern dc_com_res_id_t  DC_COM_MAGNETOMETER  ;
```

服务

表33. *dc_common.h*中的数据缓存API（服务）

```
/**
  * @brief  to register a generic cb to a given dc_db
  * @param  dc_db               data base reference
  * @param  notif_cb            user callback
  * @param  private_gui_data    user context
  * @retval dc_com_reg_id_t     created entry identifier
  */
dc_com_reg_id_t dc_com_register_gen_event_cb(
    dc_com_db_t*dc_db,
    dc_com_gen_event_callback_t notif_cb, /* the user event callback */
    void *private_gui_data);              /* user private data */

/**
  * @brief  update a data info in the DC
  * @param  dc                  data base reference
  * @param  res_id              resource id
  * @param  data                data to write
  * @param  len                 len of data to write
  * @retval dc_com_status_t     return status
  */
dc_com_status_t dc_com_write (void *dc, dc_com_res_id_t res_id, void *data, uint16_t
len);

/**
  * @brief  read current data info in the DC
  * @param  dc                  data base reference
  * @param  res_id              resource id
  * @param  data                data to read
  * @param  len                 len of data to read
  * @retval dc_com_status_t     return status
  */
dc_com_status_t dc_com_write (void *dc, dc_com_res_id_t res_id, void *data, uint16_t
len);

/**
  * @brief  send an event to DC
  * @param  dc                  data base reference
  * @param  event_id            event id
  * @retval dc_com_status_t     return status
  */
dc_com_status_t dc_com_write_event (void *dc, dc_com_event_id_t event_id);
```

表34. *dc_time.h*中的数据缓存API（服务）

```
/**
  * @brief  set system date and time
  * @param  dc_time_date_rt_info     (in) date to set
  * @param  dc_time_data_type_t      (in) time to set
  * @retval dc_srv_ret_t             return status
  */
dc_srv_ret_t dc_srv_get_time_date(dc_time_date_rt_info_t* dc_time_date_rt_info,
                                  dc_time_data_type_t time_date);

/**
  * @brief  get system date and time
  * @param  dc_time_date_rt_info     (out) date
  * @param  dc_time_data_type_t      (out) time
  * @retval dc_srv_ret_t             return status
  */
dc_srv_ret_t dc_srv_set_time_date(const dc_time_date_rt_info_t* time,
                                  dc_time_data_type_t time_date);
```

表35. *cellular_init.h*中的数据缓存API（服务）

```
/**
  * @brief  get cellular component initialization
  * @retval no return value
  */
void cellular_init(void);
/**
  * @brief  get cellular component start
  * @retval no return value
  */
void cellular_start(void);
```

版本历史

表36. 文档版本历史

日期	版本	变更
2018年6月28日	1	初始版本。

表37. 中文文档版本历史

日期	版本	变更
2018年11月23日	1	中文初始版本。

重要通知 - 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。