物联网操作系统 HelloX V1.86 发布公告

经过 HelloX 开发团队近一年的努力，在 HelloX V1.85 版本基础上，增加许多功能特性，并对 V1.85 版本的一些特性进行了进一步优化之后，正式形成 HelloX V1.86 版本。经相对充分的测试和验证之后，现正式发布。相关代码，已全部上载到 github 上（github.com/hellox-project/HelloX_OS），欢迎对 HelloX 感兴趣的朋友下载测试和试用，并进一步反馈问题。

V1.86 版本完成时间较长，主要原因是在过去的一年里，我从事的单位遭受各种外部压力。这些压力传递到员工身上，导致工作量和压力倍增，业余时间大大压缩。大多数时候都是晚上 10 点以后才下班，周末基本都要加班。即使在这样紧张的工作压力下，HelloX 的开发也没有中断，一直持续。而且对代码的质量要求更高，因为通过过去一年多的遭遇，使我认识到基础软件的重要和必要性。HelloX 属于基础软件，必须夯实基础，做到最大程度的容错和外部适应。

HelloX V1.86 主要功能简介

除继续保持 V1.85 所有特性之外，V1.86 版本主要对下列特性进行了支持或优化：

1. 用户态功能得到加强。每一个应用程序，都可以为进程形式运行，有自己独立的地址空间（32 位），可以在自己的地址空间内创建多个用户线程实现并发。通过系统调用（system call）陷入内核，访问内核功能。不同进程之间的数据和程序代码严格隔离，最大限度的保证系统整体安全性；

2. 实现相对完整的用户态代码库。引入基于 ptmalloc 开源代码的用户态堆内存管理算法，引入 ANSI 标准的 C 语言标准库，用户只需要按照通用的开发方法即可完成 HelloX 应用程序的开发，与开发 Windows、Linux 等操作系统应用没有任何不同。V1.86 版本引入了开源的 CPUID 代码库，几乎在没有修改的情况下编译成为一个用户态应用程序-cpuid.exe，可以在 HelloX 下查看 CPU 的各类信息；

3. 内核保护机制进一步增强。用户态应用程序是不可信、不安全的，为了保护内核的安全，在 V1.86 版本中引入了多种机制。最典型的就是 HANDLE 机制，即任何一个操作系统对象，比如文件、信号量、事件、互斥体等，都是以 HANDLE 方式返回给用户态，而不是直接返回这些内核对象的内存地址。用户态应用程序访问这些对象时，操作系统内核会根据 HANDLE 的值查询内部表格，得到实际的内核对象。HANDLE 机制是任何成熟复杂的操作系统必须实现的安全机制。另外一个安全机制就是用户态-内核态内存拷贝，在用户发起系统

调用时，传递到内核的是用户态内存。考虑到安全性，操作系统内核不会直接访问用户态内存，而是分配一块对应大小的内核态内存作为交换区。这种方式虽然效率上有影响，但却能够保证内核安全；

4. 实现完整的 BSD 风格的 socket 系统调用。可以在用户态调用 socket API 开发网络类应用程序，在 V1.86 版本中，内置了一个用户态的 DHCP 服务器程序，该程序通过 socket API 实现网络通信，为 DHCP 客户端分配 IP 地址。基于 socket API，后续除了一些对转发效率要求极严的网络功能（比如 IP 路由、以太网交换、IP 报文深度解析分析等），大部分网络功能都可转移到用户态，大大提升系统的伸缩性和安全；

5. 网络功能上，做了较大动作的优化。比如把 PPPoE 处理功能调度到一个单独的 CPU 核上，与 IP 路由所在的 CPU 核分离，整体提升了网络吞吐量。V1.86 版本最新支持了 Intel 82574 服务器网卡，具备了运行在高性能服务器上的基础。HelloX V1.86 的标准应用场景仍然是宽带路由器，作者重新定制了一款基于 Intel ATOM CPU 的软路由器，该路由器有 4 到 8 个基于 82574 的千兆以太网接口，整体具备大于 1G 的网络处理能力，并在家庭网络中正式使用。详细信息请参考下面章节；

6. 引入系统配置管理功能，不论是操作系统内核还是用户态应用程序，都可以把配置信息（或状态信息）写入内核维护的一个 JSON 文件中，下一次启动的时候可以直接读取。HelloX V1.86 内核集成了业界开源的 cJSON 代码，并对其做了多线程安全处理，作为系统配置管理功能的引擎；

7. 对原来版本的网络接口管理功能做了重构。在 V1.86 版本中，定义了一个新的网络接口对象__GENERIC_NETIF，即通用网络接口。所有类型的网络接口，不管是以太网、PPP 串行接口等物理接口，还是虚拟的隧道接口、VLAN 子接口，都统一到通用网络接口对象上进行管理。这使得 HelloX 的网络功能更加一致，对不同网络接口的支持也更加简便，只需要编写一个符合 generic netif 要求的驱动程序，挂载到内核中即可，HelloX 的网络管理框架可自动对其进行管理；

8. 增加了更加丰富和完整的调试与诊断手段，包括日志输出功能，异常情况下的调试信息输出功能，以及内存申请跟踪等功能。当前的内核，已经很少出现异常情况。万一出现异常，通过分析上述手段输出的信息，可以快速定位问题。

根据 github 的统计，HelloX V1.86 在原来基础上总共更新了 550 多个源代码文件，修改或增加了大约 5 万行的源代码。V1.86 版本的总代码行大约为 20 万行（不包括用户态的应用程序代码）。HelloX V1.86 内核编译后的二进制文件大小，大约在 550K 左右。

HelloX V1.86 用户态功能介绍

用户态进程功能是 HelloX V1.85 版本新引入的功能，主要目的是为了对不同的应用程序之间，以及应用程序、内核之间进行彻底的隔离，创造一个安全稳定的执行环境。是否具备用户态功能，是区分通用操作系统和嵌入式操作系统的标志之一。在 HelloX V1.86 版本中，用户态功能做了较大的增强。但用户态功能博大精深，需要考虑的因素不比内核本身要少，因此在后续版本中，还会进一步完善和增强用户态功能。

大部分嵌入式的操作系统都不具备用户态功能，在嵌入式领域，包括操作系统内核，应用程序，各种驱动程序，大多数情况都是一个解决方案厂商提供，不存在不信任的应用程序或设备驱动。而且引入进程之后，频繁的进程间切换会降低系统整体性能，因此权衡下来，嵌入式操作系统一般都不实现进程功能。

但是 HelloX 定位于物联网网关等复杂的应用场景，需要动态加载和执行第三方应用程序。如果不做安全隔离，程序之间就可以直接访问对方数据，存在巨大安全隐患。同时第三方应用质量无法保证，在没有地址空间隔离的情况下，一旦一个应用程序出问题，很大概率会导致整个系统崩溃。因此权衡下来，HelloX 最终在 V1.85 版本中引入用户态进程功能。

只是动态按需加载的第三方应用，才会以进程形式运行。HelloX 内核本身的功能，比如 TCP/IP 协议栈，USB 驱动程序，文件系统，等等功能还是在内核中运行。因为我们认为这些功能是可信任的，而且都是系统必须功能（一项功能缺失，即使操作系统内核不受影响，也已经无法提供完整的功能），因此与内核分离意义不大。

HelloX V1.86 版本的用户态功能主要包含下列这些：

1. 地址空间独立和隔离。每个进程都有自己独立的线性地址空间，在 32 位 CPU 上是 4G 大小（包括内核空间）。不同进程之间的地址空间完全独立，相同的内存地址，对不同的进程来说，完全代表不同的物理内存位置。进程的线性地址空间又进一步分为内核空间和用户空间。其中用户空间范围为 1G 到 3G 之间（共 2G），用户应用程序可以自由访问这部分内存空间，前提是明确提出内存分配请求之后。内核空间被分为了两部分：0 到 1G 之间，以及 3G 到 4G 之间。内核空间被映射到每一个进程中，应用程序代码无法直接访问，必须通过系统调用陷入内核态，才能访问内核空间。但是内核却可以自由访问用户地址空间。基于安全考虑，HelloX V1.86 版本的内核，也不会直接访问用户空间，而是经过一个内核交换区进行访问；

2. 内存保护。运行在用户态的应用程序，只能访问 1G 到 3G 之间的用户空间，无法直接访问内核空间。即使是用户空间，也必须通过一个系统调用（VirtualAlloc），

预先分配之后才能访问，否则仍然会引发异常。不同进程之间的用户空间则是完全隔离的，无法相互看到。但是 HelloX 内核也提供一个叫做 VirtualMap 的系统调用，可以把一块公共的内存同时映射到两个进程中，实现高效的进程间数据共享；

3. 应用程序动态加载。当前 HelloX 可以从外部存储介质上动态加载 PE 格式的可执行文件，并创建一个进程运行它。在加载 PE 格式的应用程序时，HelloX 会做一番详尽的格式检查，对于任何有疑问的应用程序，都会拒绝执行。一旦通过 HelloX 的检查，一个新的进程就会被创建，并启动运行。需要说明的是，从外部介质中加载应用程序，并对其进行检查和运行，是在用户态完成的。HelloX 实现了一个叫做 user agent 的模块，专门加载外部应用。内核做尽可能少的工作，把尽可能多的工作放在用户态完成。Loadapp 是 HelloX 提供的一个加载和运行用户态应用程序的工具，在 shell 界面上，输入 loadapp+应用程序路径名，即可加载并运行；

4. 系统调用功能。系统调用功能是必须实现的，在实现了用户态进程的前提下。用户态的代码无法直接访问内核，必须通过系统调用来进入内核态，才能请求内核服务。系统调用利用了 CPU 的硬件机制，通过一个固定且统一的接口，进入内核态。在执行具体的内核代码之前，系统调用的内核态部分代码会对调用参数做详细的检查，确保这个请求是合法的。比如，对于用户指定的内存地址，必须位于用户空间内，且必须已分配。如果检查失败，则系统调用会失败，同时进程会被认为有恶意行为而强制终止掉；

5. 内核对象的隐藏。在 HelloX 的内核代码中，都是通过内核对象的指针来访问内核对象的。比如等待一个互斥体（Mutex）对象，传递给 WaitForThisObject 函数的参数，就是 Mutex 对象的指针。但是让用户态也通过指针访问内核对象，则是非常危险的。虽然用户态不能直接通过指针访问内核对象，但是应用程序可以假造内核对象指针，来发起系统调用。这时候系统内核很难检查这个指针是否是非法的，如果贸然去操作，很可能因为这个内核对象不存在而导致非法操作。这就是大多数操作系统，包括 HelloX，都是以 handle（句柄）的方式来访问内核对象的。每创建一个内核对象，HelloX 内核会在进程的句柄表里增加一项，并把该项的索引值返回给用户态。用户态通过这个索引值（句柄）来发起系统调用。收到用户态的系统调用请求之后，内核会首先根据这个索引值查询句柄表，看看是否有对应的内核对象存在。如果没有，说明是一个非法的系统调用。如果有对应的内核对象，则进一步检查内核对象的状态，类型，等等。确认无误之后才会真正操作内核对象；

6. 用户态应用开发库。除了提供最基本的系统调用外，一个完整的操作系统还必须提供用户态的应用开发库，比如 C 语言标准库。C 标准库对操作系统提供的 API 做了封装，以标准化的函数对外呈现。这样在开发用户态应用程序时，程序员只需要调用 C 标准库的函数即可，无需调用操作系统的 API，使得应用程序可移植性大大增强。HelloX V1.86 支持 ANSI C89 版本的标准库函数，使用 ANSI C89 版本开发的

应用程序，可以不做修改直接移植到 HelloX。在 1.86 版本中，我们移植了很多 C 语言程序，其中比较典型的有 CPU 管理程序 cpuid 等；

7. 用户态内存管理器。实现 C 标准库函数并不仅仅是对操作系统 API 的封装，更复杂的是实现用户态的系统功能。比如对用户态内存管理（即标准 C 库函数的 malloc/free/calloc 等函数），不能直接映射到操作系统提供的 API 的，因为操作系统提供的内存分配 API 往往是大颗粒的（4K 以上），而用户应用程序所需要的大多数是一些小粒度的内存请求。这就需要在用户态提供内存再分配功能，即从操作系统"批发"内存，然后细分成更小的内存块，再"零售"给用户应用程序。这个用户态的内存管理程序并不比操作系统内核的内存管理功能简单，要充分考虑并发效率、竞争、内存使用效率等等问题。业界有很多的开源算法，HelloX V1.86 版本移植了业界广泛使用的 ptmallc 内存管理库，并做了线程安全处理，作为用户态内存分配器。

支持用户态功能，是 HelloX 由嵌入式操作系统向通用操作系统转变的标志。作为计算机领域的最基本软件，支持丰富的应用场景是操作系统得以长期存在的基础。具备高性能、实时性、高可靠的通用操作系统，必然会比嵌入式操作系统更能适应外部需求，也更容易把场景做丰富。通用操作系统与嵌入式操作系统并不是非此即彼、相互排斥的，而是可以有机统一起来。影响通用操作系统性能的传统机制，比如内存交换、延迟页面调入等，随着计算机内存的不断增大，已经变得不是很重要。把这些功能拿掉，保留用户态功能，同时对调度机制、加载机制、中断响应等做出优化，是可以有效统一嵌入式操作系统和通用操作系统的。HelloX 后续版本将超这个方向发展。

与以前版本的原则一致，在 HelloX V1.86 版本的开发过程中，我们始终坚持"稳定可靠，不留问题死角，可直接应用"为原则，所有代码都经过了详细深入的内部测试。

HelloX V1.86 网络功能简介

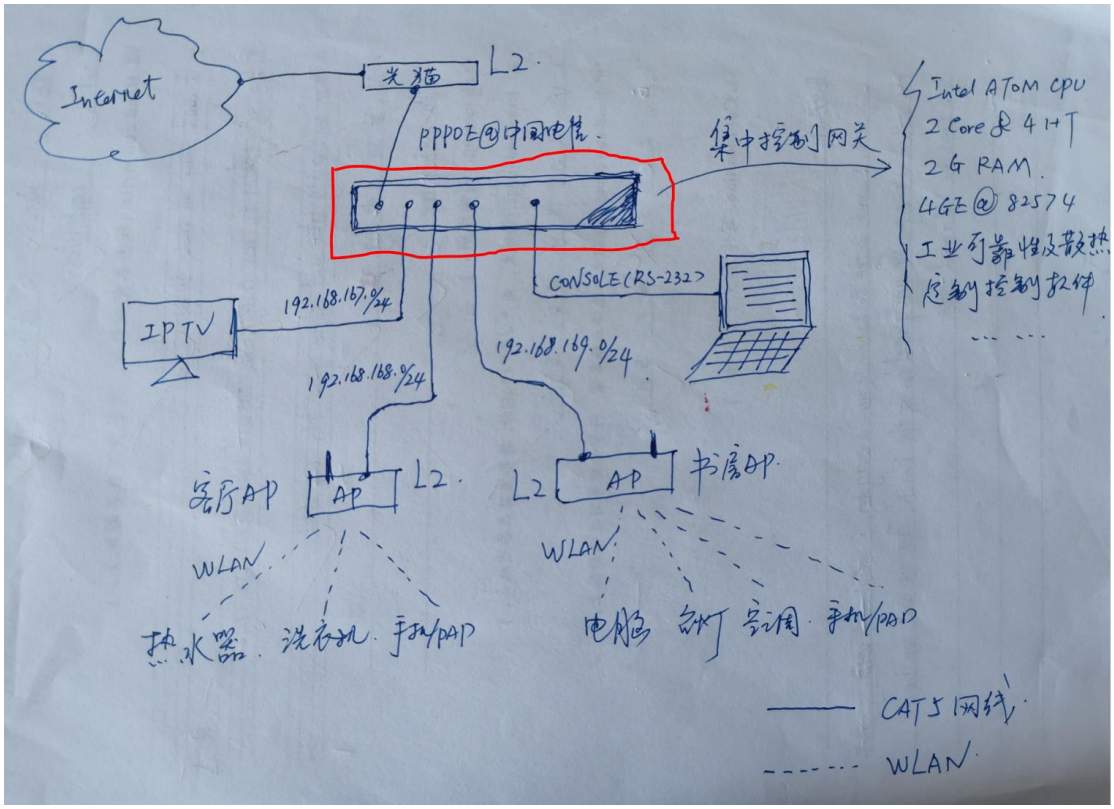
网络功能一直是 HelloX 操作系统的最重要功能。相比前一个版本，HelloX V1.86 的网络功能进一步增强，比如增加了对 Intel 服务器网卡 82547 的支持，把 DHCP Server 功能迁移到了用户态，对不同网络线程进行了多核重分配，设计并实现了统一的网络接口管理框架，等等。

在 V1.85 定制的软路由器 black box 基础上，又升级了一款新的软路由器"blue-white"，因为这款路由器的前面板有蓝、白两色。最新升级的路由器，可以支持 4 到 8 个千兆以太网接口（原来只有两个接口），可以通过 console 接口（RS232）进行配置，下图是它前面板：

所有千兆以太网接口是对等的，可以通过配置，作为 WAN 接口（连接运营商）或者 LAN 接口（连接本地交换机，或者 WLAN AP）。通过串口（console 接口）进行配置管理，个人电脑可以通过类似超级终端的软件，来控制 blue-white，如上图。

由于大多数的 WiFi 芯片都不开源，因此 blue-white 当前并不提供 WLAN AP 功能。需要外接一个二层 AP 设备，提供 WLAN 信号接入。该软路由的总体成本在 600 元人民币左右，如果批量定制，成本还可以进一步降低。但是与基于 SoC 的家用路由器相比，虽然没有功耗和成本优势，但其计算能力，则是传统家用路由器无法比拟的，这也是它作为物联网网关的重要优势。

最近刚好搬了新家，就以 blue-white 为核心，设计了一个家庭网络，并实际投入使用。整个网络的构造如下手绘图：

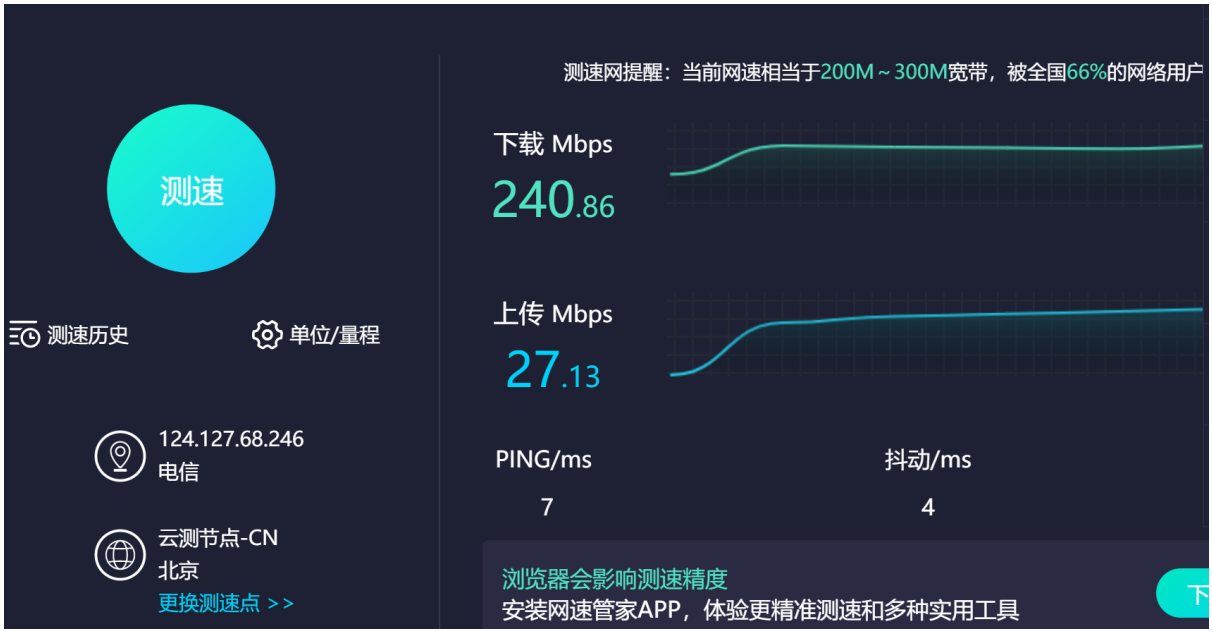


运营商提供的光猫工作在二层桥接模式，blue-white 作为整个家庭网络的核心网关，发起到运营商网络的 pppoe 连接，并为下游设备（比如 IP 电视、两个 WLAN AP 等）分配 IP 地址。家里用的家电、电脑、PAD、手机等等，都通过 WLAN AP 接入。其中 IP 电视单独通过一条物理网线接入 blue-white，这样可以把持续大流量的电视机，与突发网络流量的其它终端设备隔离开，不至于相互影响。下列是 blue-white 的一个运行截图：

```
[system-view]version
    HelloX [Version 1.860(Beta),initiated in 2019/09/09,by Garry.Xin]
    Build date:Oct  1 2020,time:11:17:29
    HelloX OS,through which you can talk to everything.
[system-view]
[system-view]loadapp c:\cpuid.exe
CPU Info:
-----------------
  vendor_str : `GenuineIntel'
  vendor id  : 0
  brand_str  : `Intel(R) Atom(TM) CPU D525   @ 1.80GHz'
  num_cores  : 2
  num_logical: 4
  tot_logical: 4
  L1 D cache : 24 (KB)
  L1 I cache : 32 KB
  L2 cache   : 512 (KB)
  L3 cache   : -1 KB
  L4 cache   : -1 KB
  L1D line sz: 64 bytes
  L1I line sz: 64 bytes
  SSE units  : 128 bits (non-authoritative)
  code name  : `Atom (Pineview)'
  features   : fpu vme de pse tsc msr pae mce cx8 apic mtrr sep pge mca cmov pat
be pni dts64 monitor ds_cpl tm2 ssse3 cx16 xtpr pdcm xd movbe lm lahf_lm
[system-view]
[system-view]network
[network_view]dhcpd list
  DHCP allocations:
  genif[3]: MAC[E4-0E-EE-83-E3-A2] -- IP[192.168.167.4]
  genif[3]: MAC[94-D0-0D-45-E3-81] -- IP[192.168.167.3]
  genif[3]: MAC[78-11-DC-0B-D1-69] -- IP[192.168.167.2]
```

CPUID 作为用户态应用程序，加载并运行后可以显示完整的 CPU 信息。

截止目前，blue-white 已运行一月有余，表现稳定。在长期外出出差的情况下，可以整体切断弱电开关，节约电量。在返回家中时，直接打开电源即可，无需做任何重新配置。在连接运营商的 PPPOE session 中断的情况下，也可以重新拨号建立连接。经过实际网络速度测试，下行速率在 230M 左右（受运营商接入带宽限制），CPU 峰值利用率不超过 50%。



在自己生活的环境中真正应用起 HelloX，并根据实际需求持续进行开发和优化，与“只开发、不使用”，或者“只使用、不开发”，感觉是完全不一样的。

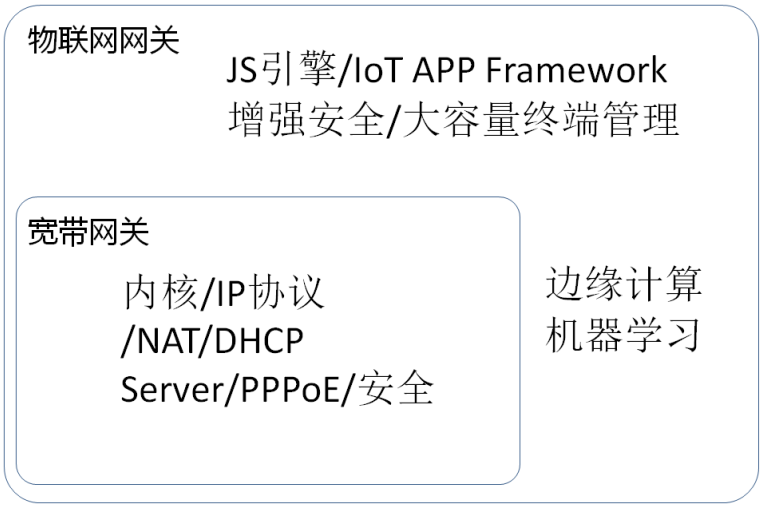
在使用过程中，随着越来越多的物联网设备接入到 HelloX 网关，我个人越来越体会到物联网操作系统的真实和价值，越来越感受到物联网网关的重要性。设想一下，物联网网关可以看到所有物联网设备的通信模式，包括报文的平均长度，发送间隔，带宽，时间分布，MAC 地址，通信协议类型，等等信息。通过这些信息，即使不对 IP 报文做深度分析（这样做是不符合法律规定的），也可以通过大数据或者 AI 技术识别出设备类型和设备能力，从而有目的的与物联网设备进行协同。单个物联网网关，是一个本地网络的控制中心，而所有的物联网网关组成的大的网络，则是整个世界的控制中心。HelloX 将聚焦物联网网关方向，做深入的技术耕耘，为人类建立安全，可靠，智能的物联网数字中心。

HelloX 进一步的开发方向

作为物联网操作系统，HelloX 将始终聚焦物联网领域的应用，为物联网量身定制一套最优的系统软件解决方案。我们认为，只有一个内核的支撑，是远远不够的。物联网和智能硬件的有效发展和壮大，需要更多技术的支持，比如人工智能，分布式计算，机器学习，等等。但一个稳定可靠和可扩展的物联网操作系统，是这些技术的最好生存土壤。

同时我们认为，物联网中的一个关键组件将会是物联网接入网关。不论是哪种应用场景，物联网网关将是物联网世界连接用户或者真实世界的最核心角色。因此，后续 HelloX 会首先瞄准这一个物联网应用场景，进行深入耕耘。物联网网关首先是一个更

安全和高效的宽带接入网关，同时具备支撑物联网应用的能力。下图示意了我们对物联网网关的理解：



物联网网关并不是孤立存在的，而是需要 AI 平台，大数据平台，物联网设备管理平台等一系列后端平台的支撑，共同组成一个面向物与物互联和协同的数字神经。我们的目标，就是构筑这样一套数字神经。

俗语有云："难事必做于易，大事必做于细"。HelloX 操作系统当前的主要应用目标，定位于物联网网关，但是要真正做出特色，做出价值，还是要从一些具体的功能入手，来打开缺口。经过与行业内一些人士交流之后，我们决定，HelloX 的下一个版本将在下列功能领域进行进一步增强和开发：

高性能的文件系统

HelloX 现有版本已经支持相对完善的文件系统，比如 FAT，NTFS（read only），裸文件系统等等。但文件 I/O 的性能还跟不上要求，当前文件 I/O 的吞吐量大约在 20M BPS 上下，而且对磁盘的访问是通过调用 BIOS 服务完成的，涉及到 CPU 模式的切换，大大影响整体效率。

在后续版本中，HelloX 要实现一个完整的基于 IDE/AHCI 接口的硬盘驱动程序，同时完整优化 FAT 文件系统，目标是使系统的文件 I/O 吞吐率能够达到 200M 以上。这样 HelloX 就具备了做网络 NAS、文件服务器等的能力。

构筑面向未来的网络协议栈

未来将对 HelloX 的网络协议栈做进一步增强和重构，开发一个业界独创，面向未来的网络协议栈。主要包括下列方面：

1. 同时支持 IPv6 和 IPv4 协议，能够按照用户的需求，同时高效处理这两种 IP 协议。我们要设计一种全新的协议架构，设计一套通用的数据结构，同时为 IPv4 和 IPv6 两种协议所用，而不是像现在的大多数实现一样，IPv4 和 IPv6 相互隔离，没有交互；

2. 进一步增强网络的安全，充分吸收新的网络架构和技术，比如 SDN 等，来做到最大限度的安全。我们认为，未来的网络安全，怎么强调都不过分。而一个自主和全新的网络架构，可以甩掉长期积累的包袱，轻装上阵，满足未来网络和信息系统的需要；

3. 在支撑网络的基础算法和数据结构有创新，满足未来网络的性能需要。比如融合 IPv4 和 IPv6 的路由查找算法，网络报文 DPI 深度解析算法等等，以期达到业界顶尖水平。

总之，我们的定位是，下一个版本的 HelloX，其网络协议栈的安全，效率，架构等方面，将达到业界顶第一的水平。

基于 JavaScript，构筑一套全新的物联网开发框架

现在有很多物联网开发框架，比如 IoTivity，三星主导的 IoT.js 等。这些框架都是基于企业开发框架来衍生或者设计的，我们认为并不能很好的适应物联网的本质特征，无法对物联网的发展启动助推作用。

我们计划，基于 JavaScript 语言，构筑一套全新的，分三个层级的物联网开发框架。通过充分的抽象和模型建立，形成一套基础的物联网模型库。基于这一套基础模型库，进一步派生出二级面向具体行业的模型库，可以成为 Tier 2 模型库。进一步地，基于 Tier 2 模型库，进一步派生出某个行业内的物联网模型。这样某个行业内的具体应用，就可以快速和直接地引用这些行业特定模型，或者对这些模型进行派生和扩展，快速高效的开发出独立于运行硬件和运行软件环境的物联网应用。

我们的站位很高，但不是盲目定位，而是基于已有 HelloX 的代码和成果，以及前期探索的基础上，做出的可以预期达到的定位。这些功能和特性开发完毕之后，将会极大增强 HelloX 的竞争力，相信能够达到物联网操作系统领域业界第一的位置。

对系统软件开发的思考与倡议

操作系统等系统软件的开发，与应用软件完全不同。系统软件要充分考虑通用性，充分考虑安全性、可靠性、性能，充分考虑各种可能的使用场景和可能的错误。系统软

件开发过程中，充分考验程序员的思维严密性，要考虑到各种可能的异常情况并做好处理。举一个例子，比如对字符串拷贝这个简单的功能，如果是作为应用软件开发，只需要做一个 for 循环，把源字符串复制到目标字符串即可。但作为系统软件处理，就需要考虑各种情况了：源字符串与目标字符串是否重叠？是否是空指针？长度是否溢出？效率如何提升？用 for 循环显然效率是最低的，要根据字符串长度、运行的目标机器等因素，给出各种优化代码，甚至嵌入汇编语言代码…。相同的代码量，系统软件的开发成本要比应用软件多两倍以上。

对于操作系统开发的难度，是随着开发的深入程度逐渐增加的。最开始时比较简单，因为有大量的资料、大量的源代码可以参考，只需要照猫画虎即可。持续一段时间，不要放弃，总能做出一个可以启动计算机的操作系统内核。但这时候，操作系统开发的难点都还没接触到。随着开发进入到深水区，真正挑战的问题逐步浮现出来：如何组织代码结构，使操作系统内核可以几乎无限量的平滑扩展？如何设计功能模块使之尽量解耦，互不影响？如何设计数据结构和算法，提升系统整体效率？如何充分利用硬件特性（比如 cache、对齐、cache 同步、内存同步、DMA 等），使得硬件效率最高？最复杂的是，操作系统的任何核心数据结构和代码，都是并行或并发执行的，如何在确保互斥的情况下，尽量减少关键区段路径，提升系统吞吐量？如何设计数据结构访问顺序，避免死锁出现？…这些问题，远比写一个网卡或者 USB 设备驱动程序要复杂。而且一旦出现内核问题，尤其是同步问题，非常难以定位，往往需要连续数天时间的思考、观察、打桩等。

更要命的是，开发操作系统短期内是无法赚钱的，而且极大概率永远不能赚钱。这就非常矛盾了：开发真正可用的操作系统，要求很高的编程水平和综合能力，但又给不了任何报酬…，因此，只有两种情况可以走下去，一种情况是背后有大公司提供资金支持，或者干脆就是大公司的顶级程序员直接开发。另外一种情况，就是开发者的情怀，全靠爱好、情怀、理想支撑下去。第一种情况在国内很少存在，因为涉及到投入产出的问题，第二种情况倒是存在不少，但大多数操作系统开发者都很清高，毕竟水平摆在那里，导致的结果就是各干各的，特立独行，甚至"相互鄙视"，无法形成规模效应。

这是操作系统等系统软件开发的现状，也是规律，个人认为短期内无法改变。但我们在了解规律的前提下，可以充分遵循规律、利用规律。HelloX 的开发不会试图违反这些规律，只会遵循规律，并试图找到在规律约束下的有效办法。但至少截至目前，还没有找到有效办法。

欢迎有志于物联网和系统软件开发的人士加入加入我们，进一步开发和优化 HelloX 操作系统。相信在我们的共同努力下，必然能够做出面向未来的基础软件平台和核心软件部件，有效促进物联网和信息化的发展水平，进一步提升人们的生活水准，为人类的发展做出贡献。

对于参与其中的朋友们来说，您的开发成果和代码输出，可以共同积累在一个相同的平台上，永远沉淀下来。如果我们做成了，您将“青史留名“。即使做不成，通过这个过程，您也可以对自身的专业技能和行业理解有一个系统的梳理和提升，必然有助于您职业的发展。打一个更加通俗的比方，相比每天拿着手机刷半天花边新闻，看半天视频，然后自责的关上手机，感叹一天又快过去了，您是不是更愿意做一些与自身专业相关且有持续积累和持续产出的事情呢？相信这样会更充实和有意义。：-)

更具体的细节，欢迎加入 QQ 群讨论：38467832

对 HelloX 和物联网操作系统的进一步信息，请关注 blog：blog.csdn.net/hellochina15