
.NET Conf China 2025
改变世界 改变自己
2025 年 11 月 30 日 | 中国 上海
C# NATIVE AOT在单片机上的应用
.NET

.NET运行时ARM(STM32)单片机移植成果

技术突破：AOT模式适配Nuttx系统

突破传统嵌入式限制，将AOT（提前编译）模式的.NET运行时深度适配Nuttx实时操作系统，解决了ARM(STM32)单片机资源受限场景下的运行时兼容性难题。

性能优势：实时性与效率双提升

相比解释型语言，AOT预编译机制消除运行时解析开销，在STM32平台实现代码执行效率做到了同级C语言的90~95%。

开发价值：简化嵌入式开发流程

依托.NET和NUTTX成熟生态，开发者可复用C#语言特性与丰富类库，无需从零编写底层驱动，显著降低ARM(STM32)单片机应用的开发门槛与周期。

生态意义：拓展.NET嵌入式边界

首次实现.NET在ARM(STM32)单片机的高性能落地，为物联网、边缘计算，机器学习等领域提供跨平台开发新选择，推动.NET向更轻量化硬件场景延伸。

C# 赋能欧标充电SECC单片机项目

01

开发周期对比：C语言8个月→C#1.5个月

客户需要增加TLS加密传输+ISO15118-20的协议，经过评估，开发时间长，所以推翻了现有C语言版本，基于C#方案重新开发，到windows正常运行1个月，nuttx适配花费半个月

02

成熟库支撑：多技术场景快速落地

C#完善的网络库（TCP IPv6/UDP）、TLS1.3加密库、序列化库（XML EXI/JSON）及P/Invoke能力，直接覆盖项目核心技术需求，无需重复造轮子。

03

多线程与异步：复杂交互高效处理

依托C#原生多线程&异步支持，轻松协调CAN总线通信、协议解析与业务逻辑，避免C语言下繁琐的线程管理与阻塞问题。

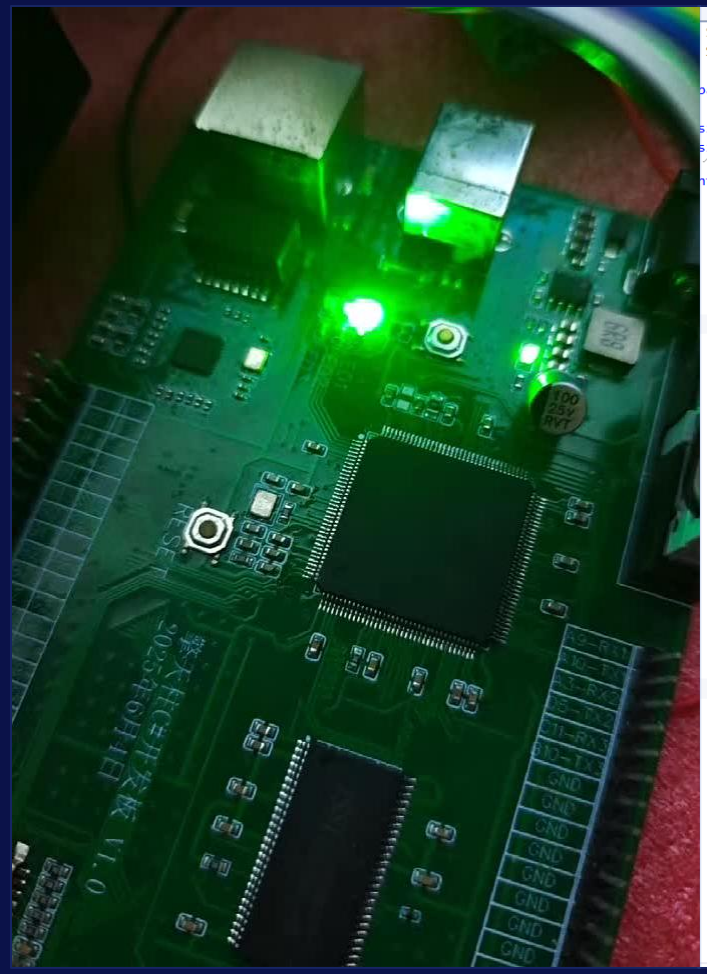
04

跨层开发优势：硬件与业务无缝衔接

通过NativeAOT实现C#与底层C代码的高效互操作，既保留单片机硬件控制能力，又利用C#的高抽象层级简化业务逻辑开发。

C#开发的LED点灯demo

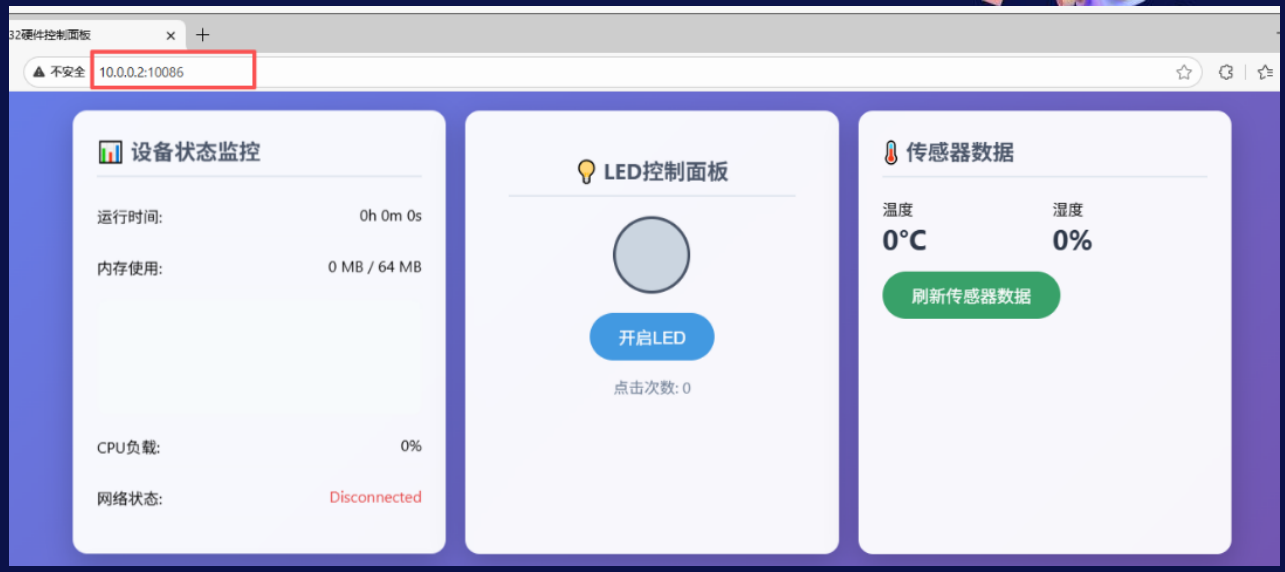
- 纯C#开发
- 不涉及PInvoke
- IOMMU映射
- 不依赖PAL层



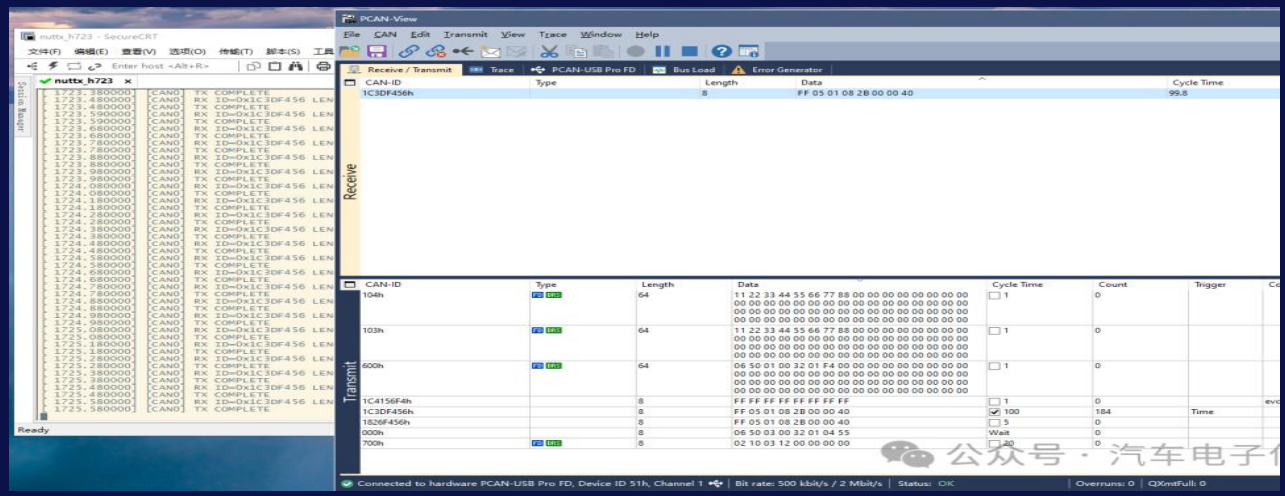
```
System;
System.Runtime.InteropServices;

pace GpioForCSharpDriver

sing uint32_t = System.UInt32;
sing uint16_t = System.UInt16;
个引用
nternal class Program

    [DllImport("*")]
    1 个引用
    internal static extern void SetIcsNativeaotActiveCode(byte[] activeCode);
    0 个引用
    static void Main(string[] args)
    {
        #region
        /*
        警告：当前所有的芯片提供的是C语言的SDK，并没有提供C#的SDK，所以在C#开发最底层驱动，
         我认为意义不大，除非你是为了学习或者研究。
        当前我们提供这个例子，主要是为了说明C#可以直接操作寄存器，从而操作硬件。
        最好还是使用C语言来操作硬件，C#层面通过open/read/write/ioctl等接口来操作硬件。
        */
        Pe3OutputInit();
        while (true)
        {
            Pe3WritePin(true);
            System.Threading.Thread.Sleep(500);
            Pe3WritePin(false);
            System.Threading.Thread.Sleep(500);
        }
        Console.WriteLine("Hello, World!");
    }


    #region
    2 个引用
    static unsafe void Pe3WritePin(bool pinState)
    {
        if (pinState != false)
        {
            GPIOE->BSRR = GPIO_PIN_3;
        }
        else
        {
            GPIOE->BSRR = (uint32_t)GPIO_PIN_3 <<(int) GPIO_NUMBER;
        }
    }
```

Kestrel demo+CANFD demo


```
.UseKestrel()
.Configure(app =>
{
    // 配置默认文件选项
    var defaultFilesOptions = new DefaultFilesOptions();

    // 设置默认文件列表，优先级从高到低
    defaultFilesOptions.DefaultFileNames.Clear();
    defaultFilesOptions.DefaultFileNames.Add("index.html");
    defaultFilesOptions.DefaultFileNames.Add("default.html");
    defaultFilesOptions.DefaultFileNames.Add("default.htm");

    // 使用默认文件中间件
    app.UseDefaultFiles(defaultFilesOptions);

    // 使用静态文件中间件，允许访问wwwroot目录下的静态文件
    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(contentRoot, "wwwroot")
        ),
        RequestPath = "", // 映射到根路径
        OnPrepareResponse = ctx =>
        {
            // 如果响应是 text/*，确保包含 charset=utf-8（避免浏览器误解编码）
            var resp = ctx.Context.Response;
            var ct = resp.ContentType;
            if (!string.IsNullOrEmpty(ct) &&
                ct.StartsWith("text/", StringComparison.OrdinalIgnoreCase) &&
                !ct.Contains("charset", StringComparison.OrdinalIgnoreCase))
            {
                resp.ContentType = ct + "; charset=utf-8";
            }
            else if (string.IsNullOrEmpty(ct))
```




```
//到了这一步已经初始化了CAN了
//编写一个简单的回显程序
var canDevice = new CanFdDevice(0);
canDevice.EventReceived+= (s, e) =>
{
    var ev = e.Event;
    switch (ev.EventType)
    {
        case CanFdEventType.CanFdEventType_RxMessage:
            {
                var rxMsg = ev.RxMessage!;
                Debug.WriteLine($"[CAN{rxMsg.CanIndex}] RX ID=0x{rxMsg.Identifier:X} DLC={rxMsg.Dlc} DATA={BitConverte
                //回显
                var txMsg = new CanFdTxMessage
                {
                    Identifier = rxMsg.Identifier,
                    IdType = rxMsg.IdType,
                    TxFrameType = rxMsg.RxFrameType,
                    Dlc = rxMsg.Dlc,
                    ErrorStateIndicator = rxMsg.ErrorStateIndicator,
                    BitRateSwitch = rxMsg.BitRateSwitch,
                    FDFormat = rxMsg.FDFormat,
                    TxEventFifoControl = 0,
                    MessageMarker = 0,
                    Data = rxMsg.Data,
                };
                canDevice.Send(txMsg);

            }
            break;
```



C# NativeAOT 最小资源占用方案

极致优化的存储需求

在C# NativeAOT的实现中，ROM空间仅需800KB(release版本,debug版本1.2M)，RAM空间控制在1MB以内。这种极致的资源优化特别适合嵌入式场景。

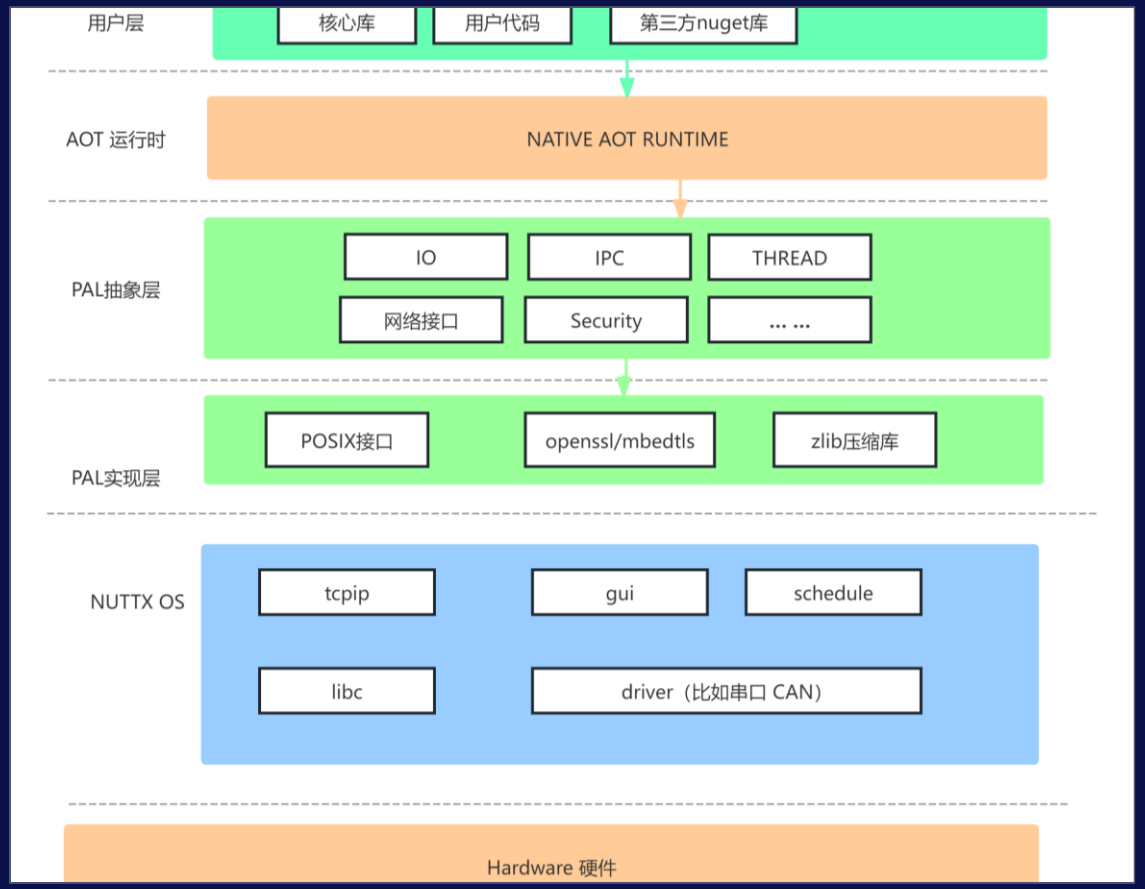
面向嵌入式的性能优势

基于NativeAOT技术生成独立的可执行文件，显著降低了启动时间和运行开销。

灵活适配多种硬件架构

当前方案以ARM M7为核心目标，但设计具有高度可移植性。未来可通过调整编译器配置支持M3、M4以及新兴的RISC-V架构。

方案核心架构



用户层的C#代码（包括您的业务逻辑、核心库和第三方库）是程序的起点。

当代码执行时（如进行文件IO、创建线程或网络通信），它会调用NativeAOT运行时来管理内存（GC）、类型系统等核心服务。

运行时需要操作系统的具体功能，它不会直接与OS交互，而是调用PAL抽象层提供的统一接口（如IO、THREAD）。

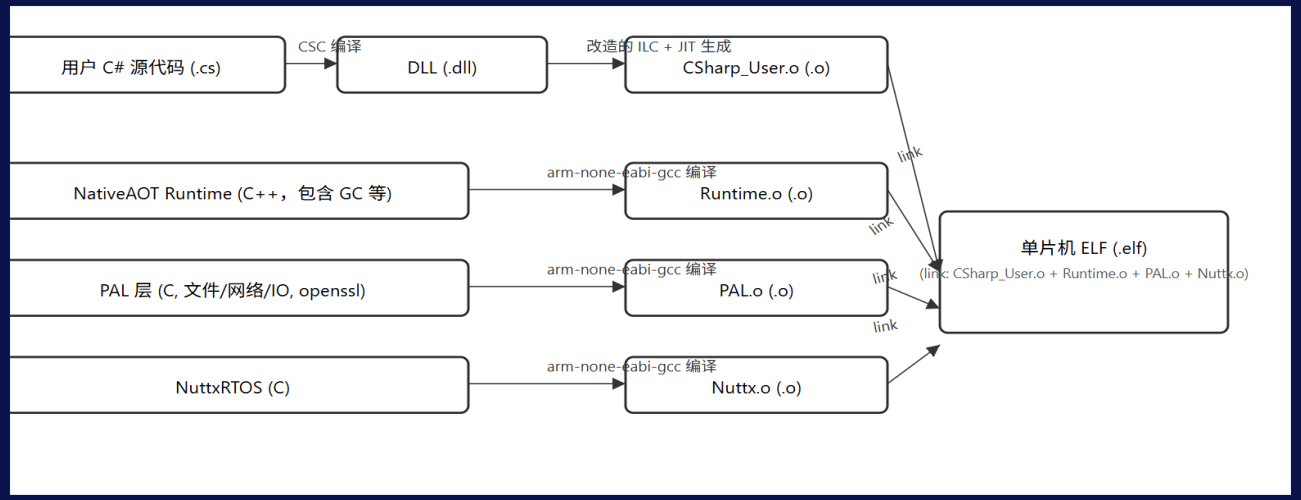
PAL实现层 是连接上层抽象世界和底层具体OS的关键。它将PAL抽象接口“翻译”成NuttX OS能理解的POSIX等标准系统调用，并可能借助openssl、zlib等底层库。

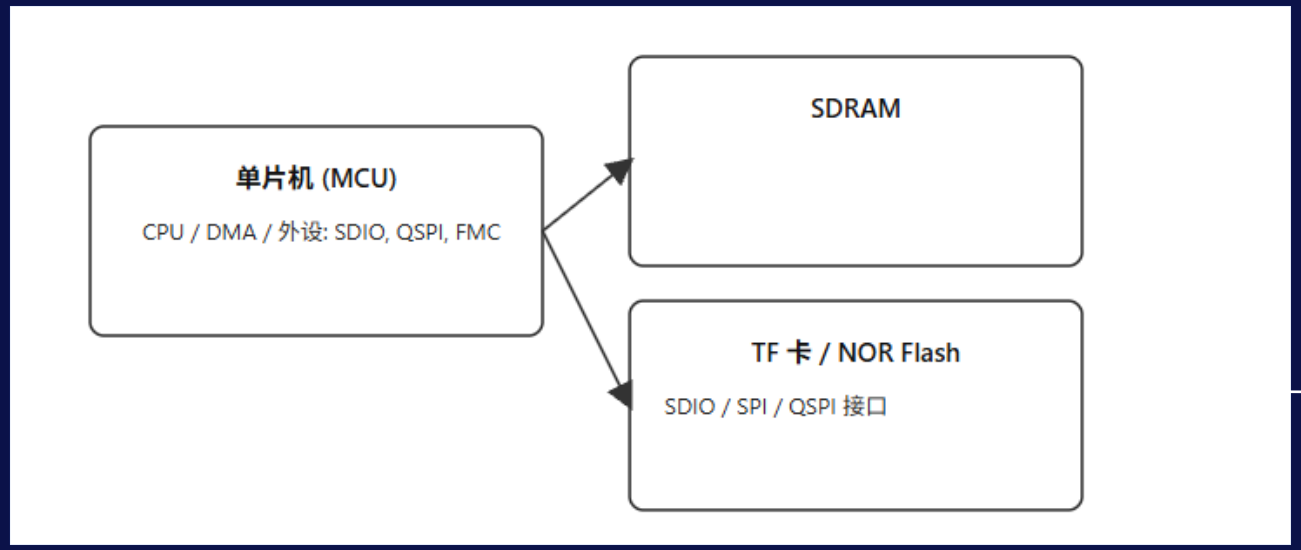
NuttX OS 接收到请求后，由其内核（schedule）进行调度，通过驱动（driver）等组件，最终在硬件上执行具体操作。

编译原理

- 核心在于通过改造的ILC+JIT工具，将C#代码（DLL）与专为单片机准备的NativeAOT运行时（C++）、PAL层（C）和Nuttx RTOS（C）的代码，统一用ARM交叉编译器生成目标文件（.o），最终链接成一个单一的单片机可执行文件（ELF）

- MCU通过专用总线与外部存储设备通信：其上方通过FMC总线连接SDRAM，用于程序运行和数据高速缓存；下方则通过SDIO/SPI/QSPI等接口连接TF卡或NOR Flash，用于程序和数据的非易失性存储





特性对比	嵌入式SOC方案 (运行Linux等OS + .NET)	NativeAOT+C# 开发单片机方案	关键差异分析
启动时间	秒级（通常1-10秒）	毫秒级（通常<100ms）	SOC需加载完整操作系统，流程长；单片机常为裸机或轻量RTOS，启动接近“上电即运行”。
视频处理能力	强大。通常集成GPU、视频编解码器等专用IP核，支持复杂图形界面和高清视频播放。	较弱，主要驱动简单显示屏，处理能力集中于控制。	SOC为复杂多媒体应用设计，硬件集成度高；单片机目标场景无需此能力。
串口(USART)最高速率	常见约2-3 Mbps。虽硬件潜力或更高，但在Linux等系统下，驱动和系统调度会限制实际稳定速率。	可达8-12 Mbps。程序直接操作硬件，中断响应及时，可实现更高的稳定波特率。	单片机实时性更强，软件开销小，更适合高速、确定性串行通信。
CAN总线性能	可能存在瓶颈。集成控制器或为USB等高速外设共享资源；在通用OS下，协议栈和系统负载可能导致延迟或丢帧。	可靠性高。普遍采用成熟控制器（如博世）；应用更接近硬件，可做到满载且不丢帧。	单片机架构更契合高实时性、强时序要求的工业总线通信。
SPI最高速率	通常在十几至几十Mbps。受系统调度、驱动缓冲区等因素影响，难以发挥硬件极限速度。	可达50-100 Mbps。程序通常可直接配置外设时钟，实现极高的数据传输速率。	原因同串口，单片机在简单高速同步串行通信上具有天然优势。
工作结温范围	商业级（0℃ ~ +70℃）常见，工业级（-40℃ ~ +85℃）选项价格显著提升。	更宽。广泛提供工业级甚至汽车级（-40℃ ~ +105/125℃）产品，原生适应恶劣环境。	单片机设计针对工业控制，强化了耐温特性；SOC集成度高，对高温更敏感。
芯片单价	相对较高，尤其高性能或多核型号。	更具价格优势，在相同温度级别下尤其明显。	SOC因集成复杂IP核和先进工艺，成本更高；单片机结构简单，成本效益好。
整体BOM成本	较高。需外置DDR内存、PMIC（电源管理芯片）、更多无源元件，PCB层数多，设计复杂。	较低。多为单电源供电，无需PMIC；芯片常内置Flash和SRAM，可省外部存储器；PCB简单。	

未来技术拓展规划：应用移植与芯片支持

核心GUI框架移植计划

重点推进LVGL Sharp、Avalonia等GUI框架移植，覆盖嵌入式系统开发场景，提升界面交互能力与开发效率。

多架构芯片支持布局

全面支持ARM M3/M4/M55等，和RISC-V架构单片机，兼顾传统与新兴指令集，强化硬件兼容性。

技术拓展目标定位

通过应用与芯片双维度拓展，覆盖更广泛硬件平台，满足工业控制、智能设备等多样化场景需求。

预期生态影响展望

推动嵌入式开发灵活性提升，加速跨平台应用落地，助力构建开放、兼容的嵌入式技术生态。

.NET Conf China 2025

改变世界 改变自己

THANK YOU