Open Source 西门子

Beijing · 2026

Oct. 29
Industrial Open Source

Oct. 30
Open Source Robotics

Agenda

中科时代
SINSEGYE

面向具身机器人的 Xenomai

Xenomai for Embodied Robots

中科时代(**深圳**)计算机系统有限公司

演讲人：宋健玮 刘杨（中科时代 Sinsegye）

2026.10.30 · Open Source @ 西门子

中科时代
SINSEGYE

目录

1. 中科时代公司简介
2. 机器人行业案例
3. 具身机器人为什么需要MetaOS硬实时
4. 社区实战：一个热中断补丁的诞生

刘杨 688181

PART

1

中科时代公司简介



刘杨 688181

中科时代（Sinsegye） 公司简介

中科时代 SINSEGYE

成立时间	员工规模	获得融资
2021 年	200 +	B2 轮

中科时代（深圳）计算机系统有限公司是中国科学院计算所继寒武纪、海光信息、曙光信息、龙芯中科、联想等之后的又一计算机体系结构的创新性产品驱动型科技企业。公司深耕高端自动化与工业 AI 智能体，市场份额稳居工业自动化一级市场前列，产品已广泛应用于风电、半导体、机器人、具身智能、流程工业等关键领域。

公司秉承“**基于计算机技术与 AI 的工业智能化**”的发展理念，拥有工智机、MetaOS 双域操作系统、MetaFacture 混合编程环境、MetaCore 运行时、工业 AI 算法软件、远程 IO 和伺服 & 电机等核心产品。

面向工业场景，中科时代**致力于提供高效、稳定、可靠的智能一体化产品与控制解决方案**，通过深度融合自动化、计算机与 AI 软件技术，打造专业、开放架构的工业智能计算机产品与技术服务体系。

马君

中科时代创始人兼 CEO



复合经验 拥有智能产品、计算机体系结构、算法、销售市场经验，精通产品创新与团队人才引进。

丰富履历： 曾任中石油 BGP 数据科学家、B2C 电商 startup 联合创始人、阿里巴巴达摩院决策智能实验室算法专家、工业 AI 芯片传感器联合创始人，项目获 BV、经纬等一线资本投资。

科研背景： 中国科学院计算所博士，清华博士后、杜克大学访问学者；发布 5 篇 CCF A 类论文，授权专利 28 项。

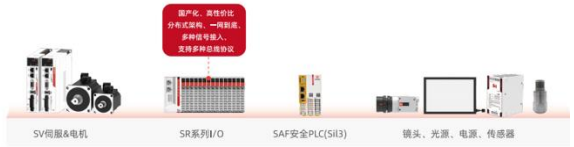
▸ 嵌入式工智机（EPC）



▸ 标准型工智机（IPC）

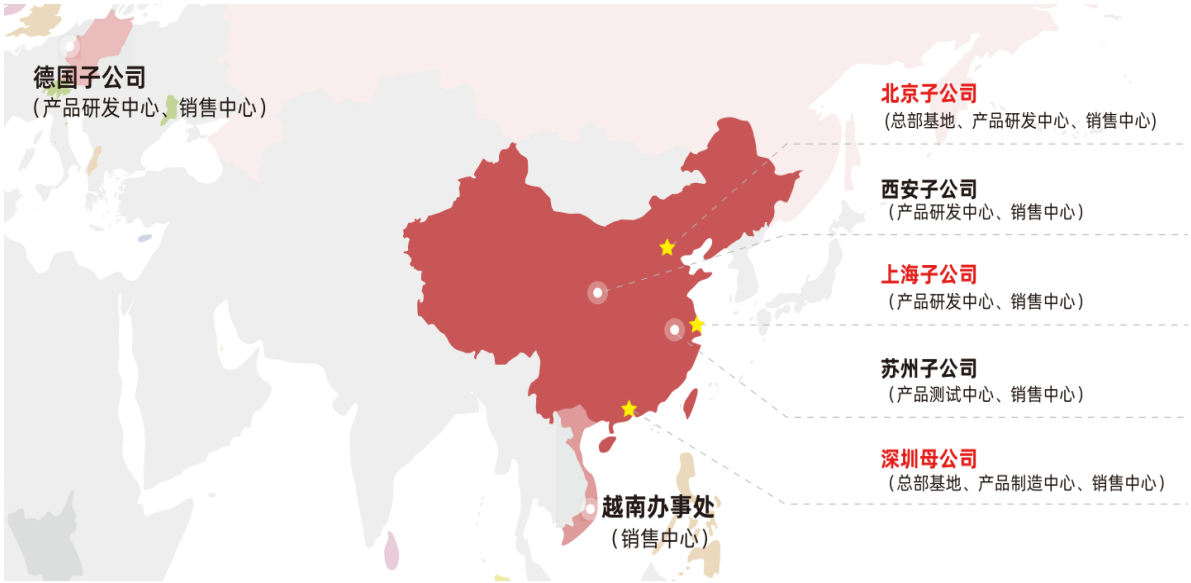


伺服&电机、远程IO、其它组件产品



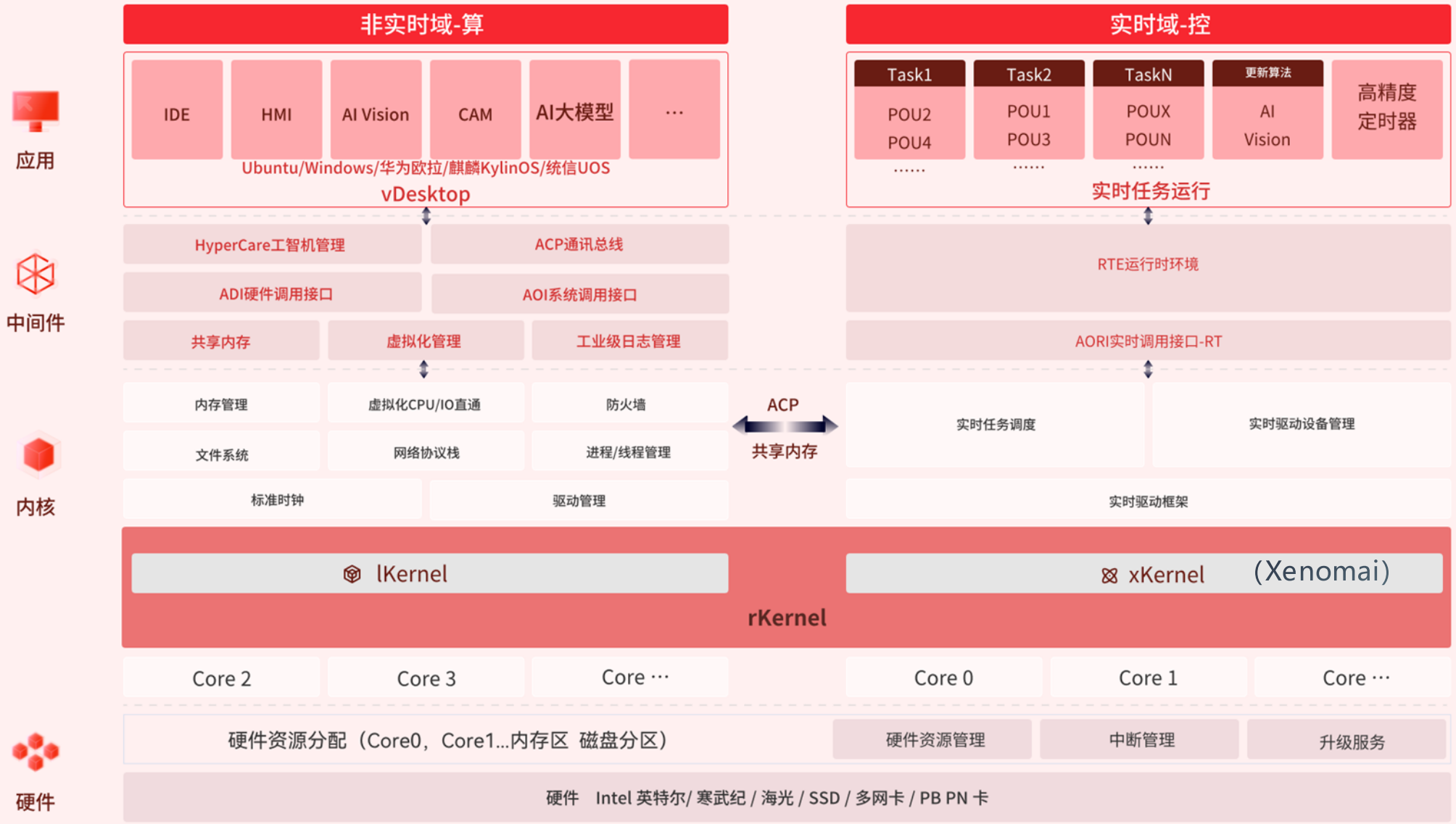
iAutomaiton 自动化行业软件





MetaOS：基于Xenomai的Linux发行版

中科时代 SINSEGYE



PART 2



机器人行业案例



刘杨 688181

MetaOS + SP8000 具身机器人案例



Kernel 6.12 + Xenomai 3.3.1

可搭载能英伟达RTX4060m 8G
用于推理加速

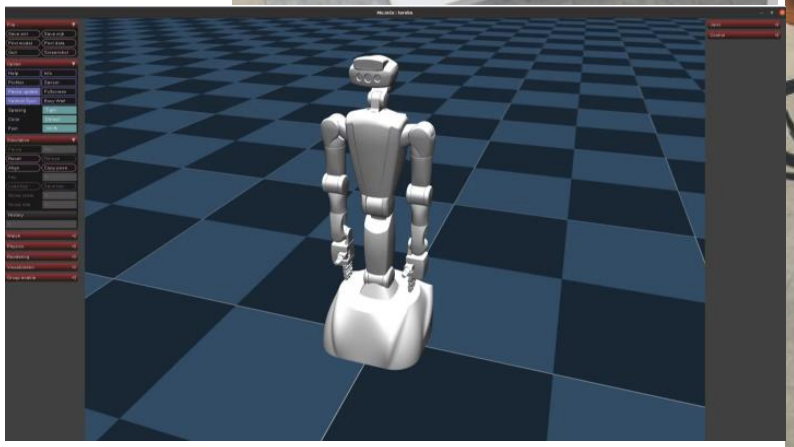
产品名称	SP8000-C340	SP8000-B440
操作系统（可选）	Meta OS	
处理器	Ultra5 125H，TDP 28W	Ultra 7 255H，TDP 28W
内存	DDR5 32GB	DDR5 64GB
存储	M.2 NVME 2280 1TB	
显示	1xType-C，Support DP ALT and DC OUT 5V@3A	
网络	1xRJ45：Intel i226 10/100/1000Mbps 4xType-C：Intel i210 10/100/1000Mbps	
I/O	2xRS485 2xCAN FD，支持最大5M传输速率	
USB	Type-C1/DP&Type-C2&Type-C3&Type-C4	
无线通讯	AX210 网卡	

中科第五纪的自研模型和算法运行在非实时域。

- 核心模型 FAM-1.3 - 超少样本模型，只用传统大模型 1% 的数据 ---- 赋能手上
- 导航模型 FiveAges-Nav，在机器人顶会 RSS 2026 国际挑战赛上，11 项指标全部拿了全球第一， --- 赋能脚下
- 世界模型BridgeV2W做 "大脑预判" - 整体空间预判

别人家的机器人换个料箱、换个场地就得重新编程调试，中科第五纪的不用！





MetaOS已适配 MuJoCo数字孪生



MetaOS 具身机器人软件生态分层架构

非实时域-算 · 智能感知与仿真训练

应用层 | 算法与仿真交互

- 仿真环境：Isaac Sim / Gazebo / MuJoCo (数字孪生训练)
- 智能感知：Isaac ROS / OpenVLA (大模型具身理解)
- 调试工具：Foxglove Studio (多端数据可视化监控)
- 运动规划：MoveIt 2 (机械臂逆解与轨迹生成)
- 自主导航：Nav2 (SLAM与动态避障)

中间件层 | 算力调度与框架

- 分布式通信：ROS 2 Humble / Dora-rs (异构节点调度)
- 决策引擎：RoboNeuron (端到端具身智能推理框架)
- 资源管理：容器化部署与多进程并发调度

内核层 | 通用算力基座 (IKernel)

Linux Kernel v6.12深度定制，多线程并发，GPU/CUDA算力，高吞吐数据处理能力。

标准 Linux 内核

POSIX兼容，生态极其丰富，CUDA 、 国产海光DCU

ACP 共享内存

实时域-控 · 运动控制与执行调度

应用层 | 控制与规划执行

- 动力学：Pinocchio (高精度刚体动力学计算)
- 电机 1kHz+ 闭环控制
- 实时轨迹插补

中间件层 | 实时调度环境 (RTE)

- 任务管理：硬实时任务调度器 (微秒级抢占)
- 总线通信：EtherCAT / CANOpen (CAN/CANFD) 工业总线适配
- 同步机制：硬件时基同步 (PTP精准时钟协议)

内核层 | 硬实时微内核 (xKernel)

微内核，纳秒级响应、确定性调度，为机器人关节控制与运动执行提供硬实时保障。

RT 抢占式调度器

微秒级上下文切换，无锁设计

RTDM 实时驱动

绕过内核，直接映射硬件IO

工智机 底层硬件基座 (Hardware Foundation)： 异构计算平台

x86 CPU + NVIDIA GPU/Tensor Core + PCIe 4.0 高速总线 + NVMe SSD + DDR5 高带宽内存 + 工业级实时网卡



已适配 ROS2 — 机器人操作系统中间件

01 核心定位与适配现状

分布式通信基座：基于DDS标准构建，提供节点发现、话题/服务通信及参数管理，是连接感知、决策与执行的标准化中间件。

桥接适配 ROS2 无原生 Xenomai 内核支持，需通过开源桥接方案实现跨域交互，本体运行于非实时域。

部署策略：ROS2 运行在标准 Linux 用户空间处理逻辑，高实时性控制算法剥离至 Xenomai 实时域独立运行。

02 跨域通信与架构隔离

XDDP 低延迟管道：利用 Xenomai 提供的跨域数据管道 (XDDP)，实现 Linux 域 ROS2 节点与 Cobalt 实时线程间的高效、低延迟数据收发。

稳定性边界：ROS2 不进入实时内核空间，避免了复杂框架对实时系统确定性的干扰，保障控制安全。

版本演进：ROS1 曾短暂支持 Xenomai 后因维护成本移除；ROS2 采用更灵活的桥接模式，解耦设计更稳健。

ros_xenomai_bridge

由 MasterERTS 维护，基于 XDDP 的标准通信桥接库，实现 ROS2 话题与 Xenomai 实时线程的双向透传，是生态基石。

实时框架

Brambovich 开源的 20sim-Xenomai 项目，专为树莓派优化，集成了从仿真模型到硬件IO的实时调度流水线。

差速机器人实战

ROS2-Xenomai-RTOS 实战项目，演示了在差速驱动机器人上如何结合 ROS2 的高扩展性与 Xenomai 的实时控制闭环。

适配中 Dora-rs — 数据流导向机器人框架

核心架构与性能优势

- **纯 Rust 构建：**内存安全且并发高效，专为实时数据流场景设计，无 GC 停顿风险。
- **极致零拷贝：**基于 Apache Arrow 共享内存消息，性能比 ROS2 Python 快 10-17 倍。
- **多语言融合：**原生支持 Rust、Python、C/C++ 节点混合开发，兼顾效率与灵活性。

Xenomai 适配现状与机制

当前状态：与Xenomai社区共同维护

Dora-rs 的实时性源于架构层面的零拷贝与无锁设计，Rust 运行时对接 Xenomai 调度进入 Cobalt 硬实时域。

硬实时场景部署建议

若需实现微秒级硬实时控制，建议将强实时逻辑剥离为独立的 Xenomai Cobalt 实时任务，通过共享内存（Shared Memory）或 FIFO 管道与 Dora-rs 节点进行低延迟数据交互，从而构建“非实时计算 + 硬实时控制”的解耦式高可靠架构。

两种产品部署形态：同一套工智机，纯 Linux 或 Linux + KVM(Windows 10)

实时域始终独占隔离核、不被虚拟机影响；区别只是非实时生态跑在哪里

形态 A • 纯 Linux

Cobalt 实时域 + 完整 Linux 域同机共存
软件栈：Ubuntu + ROS 2（MoveIt2、Nav2、Gazebo 仿真、状态估计、上位机界面）
开源机器人全家桶，算法迭代快、社区资源丰富
单系统、启动快、资源利用率最高
适用：机器人原生 Linux 软件栈、对启动时间和算力利用率敏感的场景

形态 B • Linux + KVM(Windows 10)

Windows 10 虚拟机跑上位机、视觉、商用组态
软件栈：TwinCAT / CODESYS 软 PLC · Halcon / VisionPro 机器视觉
LabVIEW 测试测量 · 商用相机 / 激光雷达 SDK 多为 Windows 版
Linux + Cobalt 实时域独占隔离核，虚拟机不碰实时核
适用：客户已有 Windows 生态、现场必须跑商用上位机 / 视觉软件的项目

· 两种形态同一套 SP8000-C340 工智机同时兼容，按客户现场软件栈选配，实时确定性不打折扣。

刘杨 688181

PART 3



具身机器人实时性指标



刘杨 688181

为什么具身机器人需要MetaOS硬实时加持?

控制环延迟要求

- 电机电流环：**10-50μs**
底层执行单元，抑制电机干扰的极速响应
- 关节伺服环：**100μs-1ms**
保障机械臂精准轨迹跟随与动态平衡
- 运动规划层：**10-100ms**
负责全局路径规划与行为决策调度

标准 Linux 的局限

(Ubuntu, 欧拉，统信，开源鸿蒙)

- 调度不确定性：**毫秒级抖动**
远超微秒级控制需求，破坏系统稳定性
- 内核抢占限制：
非抢占式内核段导致不可预测的阻塞延迟
- 资源竞争风险：
通用进程调度难以保障控制任务的最高优先级

MetaOS的硬实时价值

(基于 Xenomai)

- 双内核架构：**微秒级响应**
Cobalt 微内核提供确定性硬实时保障
- 域隔离机制：
实时域与 Linux 非实时域并行，互不干扰
- 生态无缝兼容：
复用 Linux 丰富的驱动与软件生态资源

典型工业机器人控制频率基准

稳定运行门槛：**1kHz**高频闭环控制频率

高动态控制的严苛抖动指标

确定性要求：**< 10μs**最大允许调度抖动

刘杨 688181

机器人实时软件栈：控制环独占隔离核，非实时任务跑在 Linux



实时域：可抢占 Linux
· 固定优先级调度
· 关中断抖动隔离
· 最坏延迟可预测

· 把控制环钉在隔离核上，配合 grub 参数（isolcpus / nohz_full / irq affinity），彻底与 Linux 非实时调度隔离。
· Linux 非实时域继续享受完整的生态，实时与非实时各得其所——这就是机器人上"一套系统、两个世界"的落地形态。

刘杨 688181

实时通信：关节指令必须在确定时刻、以可预测延迟送达

现场总线是控制环与关节驱动器之间的确定性通道，收发必须和控制环跑在同一个实时域里。

周期确定

每帧在精确时刻发送，不被 Linux 调度抖动带偏

低延迟低抖动

收发路径全程在 Cobalt 域，最坏延迟可预测

多关节同步

多轴指令同一时刻下发，保证协调运动一致性

同域闭环

收发与控制环同优先级，闭环不跨域

- 普通 socket / 网络栈收发：缓冲、调度、NAPI 带来不可预测延迟，不能用在关节指令通道上。
- 实时总线上的收发走 RTDM 实时驱动，与控制环同处 Cobalt 域，逐帧时序由实时内核保证。
- 这一层做对了，上层 1 kHz 控制环算出的力矩，才能干净、准时地送到每一个关节驱动器。

刘杨 688181

参考：RTDM 实时驱动接口 · Xenomai 现场总线实时化实践

机器人关节--通讯总线对比

RS485（Dynamixel）｜CAN FD（CANopen FD）｜EtherCAT（CoE + DC 同步）：带宽 · 闭环周期 · 同步抖动 · 成本与拓扑取舍

对比维度	RS485（Dynamixel）	CAN FD（CANopen FD）	EtherCAT（CoE + DC 同步）
总线带宽	最高 **4~10 Mbps**，半双工	仲裁段 1 Mbps，数据段 **2~8 Mbps**；单帧最大 64 字节	**100 Mbps** 全双工，带宽利用率 > 95%
稳定闭环周期（6 轴）	**50~200 Hz**；极限勉强 500 Hz，抖动失控	**500~1000 Hz**，1 kHz 稳定闭环	**1 kHz~10 kHz**，250 µs 周期可稳定运行
多轴同步抖动	**±2~8 ms**；无硬件同步，纯轮询，时差大	**±0.5~2 ms**；优先级仲裁，无全局时钟	**±1~5 µs**；DC 硬件时钟，全轴同一时刻采样执行
主站系统要求	普通 Ubuntu / Windows，**无需实时内核**	建议 **RT-PREEMPT**；普通系统抖动放大至 3~5 ms	必须 **Xenomai / RT-PREEMPT** 硬实时；普通桌面抖动破百微秒
单关节硬件成本	**极低**：普通 MCU + RS485 收发器，BOM 数元	**中等**：MCU + CAN FD 控制器，无专用芯片	**高**：须 ESC 芯片 ET1100/ET1200 或 FPGA 软核，成本上浮约 80%
拓扑规模	多节点总线，最多 **32 轴**	多节点总线，单网段最多 **110 轴**	菊花链 / 树形，支持 **上百轴**，布线灵活
选型定位	**低成本 · 位置采集** 模仿学习示教采集、慢速抓取摆放	**折中 · 1 kHz 闭环** 自研桌面臂 / 人形上肢，位置 + 简单力矩闭环	**硬实时 · 力控首选** 全关节力矩闭环、阻抗控制、柔性交互、力反馈遥操作

抖动 = 多轴同步误差，闭环周期 = 下发指令 + 读取位置/力矩整轮周期；EtherCAT 开启 DC 分布式时钟、运行于硬实时主站，数值随拓扑与负载变化。一只机器人手臂：约7轴+1灵巧手

实测验证：整机满载 24 小时，控制环最大抖动约 10µs

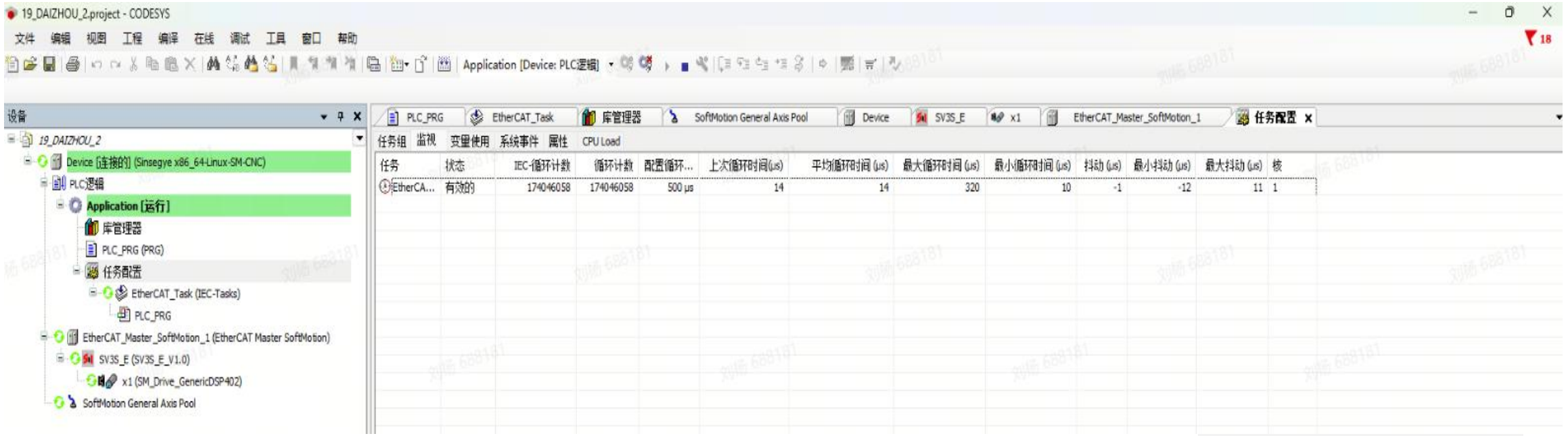
CODESYS EtherCAT 500µs 带单轴跑在隔离实时核上，与 CPU / GPU / 磁盘 / 内存压力并行，连续 24 小时

实时负载
CODESYS EtherCAT 500µs 周期 · SoftMotion 单轴（SV3S 伺服）

背景压力
stress -c 12 -d 6 -i 6 --vm 10
glmark2 --run-forever
gpu_burn

连续时长
24 小时连续运行 · 1.74 亿个控制周期无丢帧

最大抖动
约 10µs（CODESYS 任务监视实测：平均 14µs 循环）



来源：CODESYS EtherCAT_Task 任务监视 · SP8000-C340（Intel Core Ultra 5 125H + NVIDIA RTX 4060）· MetaOS 2.0（Linux 6.12 + Xenomai 3.3.1）

PART 4

热中断补丁实战





刘杨 688181

问题现象：在什么平台、什么场景下出的问题

整机冻结而非单环抖动——实时栈长时间运行 + 高频通讯后暴露

平台与版本	异常表现	触发条件	第一反应
Intel Core Ultra 5 125H + NVIDIA RTX 4060 SP8000-C340 工智机	运行一段时间后整机画面冻结 鼠标键盘均无响应	CANFD 通讯产生大量中断 高频场景更容易触发	先怀疑核显 i915 排查后转向中断路径

- 现象是整机级冻结（画面卡死、键鼠无响应），不是单个控制环抖动——指向系统级调度 / 中断路径。
- 关键背景：MetaOS 2.0（中科时代工业实时 OS）基于 Linux 6.12 开启 Dovetail 管道、搭载 Xenomai Cobalt 3.3.1，隔离核 + tickless idle 停 tick 正在生效。
- 第一时间先怀疑核显（i915）——它同时驱动显示与 GPU 工作队列；但真正的线索在中断侧。



SP8000-C340 工智机（现场故障平台）

来源：中科时代 SP8000-C340 现场复现记录 · Intel Core Ultra 5 125H + NVIDIA RTX 4060 · MetaOS 2.0（Linux 6.12 · Xenomai 3.3.1）

问题是怎么被发现的：从现象到怀疑

排除法锁定 dovetail 管道 → 缩短 RCU stall 超时加速复现 → trace-cmd + NMI backtrace 取证

先怀疑核显
画面冻结最像显示 / GPU 侧
换 xe 驱动 · 定频核显 · 关 media 均无效

→

分水岭在补丁
同配置不打 Xenomai：跑一天不卡
只打 dovetail 仍卡 · KCSAN: handle_oob_irq

→

加速复现
RCU stall 超时缩短至 5 s
让卡死快速现形

→

抓取铁证
trace-cmd 采集 cobalt / sched / irq / rcu
NMI backtrace：卡在 timer 锁上

关键证据 • 一次 trace-cmd 抓到的完整链条

rcu: INFO: rcu_preempt detected stalls on CPUs/tasks —— CPU 10 卡死，t=10017 jiffies，ncpus=12
NMI backtrace：kworker/10:0 停在 native_queued_spin_lock_slowpath，调用链指向 thermal 中断 → timer 锁
结论：Kernel panic - not syncing: RCU Stall —— 系统彻底冻结，无法交互

- 排除法缩小包围圈：不打 Xenomai 补丁不卡、只打 dovetail 仍卡 → 问题就在 dovetail 管道层，KCSAN 同时爆出 handle_oob_irq 数据竞争。
- 缩短 stall 超时是「放大镜」：把默认数十秒才暴露的卡死压缩到 5 s 内现形，配合 CANFD 高频中断快速复现。
- 热环境线索：24V 供电整机 CPU 冲到 103°C，/proc/interrupts 里 TRM 热中断计数每秒涨上百——热中断才是真实高频源。

刘杨 688181

来源：现场 trace-cmd 抓取日志 · -e cobalt* -e sched: -e irq: -e irq_vectors: -e signal: -e rcu:rcu_stall_warning

关键证据：三组日志锁死结论

KCSAN 数据竞争 → RCU Stall 持续 34 分钟 → NMI backtrace 卡在 timer 锁 → 温度铁证，四条线收敛到同一个缺口

KCSAN 数据竞争 • 7/14

只打 dovetail 仍卡死：handle_oob_irq 对同一寄存器被 CPU10（idle 路径）写、CPU7（futex 唤醒）读
value 0x14000 → 0x4000
oob 中断路径无锁保护，问题在管道层

RCU Stall 马拉松

同一颗 CPU7 不推进，stall 每 30 s 重复：
t=60017 → 2040535 jiffies
持续约 34 分钟
RCU kthread 被饿死，iwlwifi 连带崩溃

NMI backtrace 自旋点

thermal 中断 → therm_throt_process → add_timer_on → lock_timer_base → spin_lock_slowpath
对端 kworker 持锁走 tmigr_handle_remote：
硬中断上下文自相矛盾

温度铁证

24V 供电整机，CPU2 冲到 103°C CRITICAL
TRM 计数每秒最多 +318 次，80–103°C 反复跳变
高频热中断是常态流量，不是偶发

关键证据 • 三条日志链如何收敛

KCSAN 指向 dovetail 管道层 → NMI backtrace 指向 timer 锁 → 温度指向高频 Thermal 向量，三条独立证据线收敛到同一个缺口：THERMAL_APIC_VECTOR 未被管道化。
为什么找了一个月才现形：必须热负载 + 实时停 tick + GPU 负载同时成立；把 RCU stall 超时从默认几十秒压到 5 s 后才稳定复现。

- 对照实验是分水岭：不打 Xenomai 补丁跑一天不卡、只打 dovetail 仍卡 → 问题不在 Cobalt 域，而在 dovetail 管道向量分发。
- trace-cmd 一次性挂了 cobalt* / sched / irq / irq_vectors / rcu:rcu_stall_warning 五组事件，NMI backtrace 才抓到自旋现场。
- 反直觉点：画面冻结先指向核显，实际 i915 只是「持锁路过」的倒霉蛋，温度才是真凶。

刘杨 688181

来源：2026/7/14 KCSAN 日志 · RCU stall 现场 dmesg · trace-cmd NMI backtrace · 8/7 温度监控脚本 · MetaOS 2.0

关键证据 · 定位现场：NMI backtrace 原文

热中断在 IRQ 上下文抢 timer 锁，持锁的恰是同 CPU 的 tickless idle 停 tick 路径——正面相撞，一次抓全

```
[ 1815.941603] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
[ 1815.947739] rcu: 4-...!: (73 ticks this GP) idle=e5cc/1/0x4000000000000002 softirq=71359/71361 fqs=40
[ 1815.957083] rcu: 5-...!: (85 ticks this GP) idle=63c4/1/0x4000000000000000 softirq=70592/70594 fqs=40
[ 1815.966417] rcu: (detected by 8, t=65002 jiffies, g=108849, q=693 ncpus=12)
[ 1815.973503] Sending NMI from CPU 8 to CPUs 4:
[ 1815.973510] NMI backtrace for cpu 4
[ 1815.973515] CPU: 4 UID: 0 PID: 0 Comm: swapper/4 Tainted: G U W OE 6.12.90-13-rkernel-cip24-x64-industry #metaOS
[ 1815.973519] Tainted: [U]=USER, [W]=WARN, [O]=OOT_MODULE, [E]=UNSIGNED_MODULE
[ 1815.973520] Hardware name: Sinsegye SP8000-C340/SP8000-C340, BIOS SP80000020260708B 07/08/2026
[ 1815.973521] IRQ stage: Linux
[ 1815.973522] RIP: 0010:native_queued_spin_lock_slowpath+0x78/0x380
[ 1815.973531] Code: 2c 24 08 0f 92 c2 41 8b 04 24 0f b6 d2 c1 e2 08 30 e4 09 d0 a9 00 01 ff ff 75 5c 85 c0 74 14 41 0f b6 04 24 84 c0 74 0b f3 90 <41> 0f b6 04 24 84 c0 75 f5 b8 01 00 00 00 66 41 89 04 24 5b 41 5c
[ 1815.973532] RSP: 0018:ffffb065802dce38 EFLAGS: 00000002
[ 1815.973535] RAX: 0000000000000001 RBX: ffff9334dfaaa428 RCX: 0000000000000000
[ 1815.973536] RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffff9334dfa53580
[ 1815.973537] RBP: ffffb065802dce58 R08: 0000000000000001 R09: 0000000000000000
[ 1815.973537] R10: 0000000000000000 R11: ffffb065802dcff8 R12: ffff9334dfa53580
[ 1815.973538] R13: 0000000000000000 R14: 0000000029f00004 R15: ffff9334dfa53580
[ 1815.973539] FS: 0000000000000000(0000) GS:ffff9334dfa00000(0000) knlGS:0000000000000000
[ 1815.973540] CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 1815.973541] CR2: 00007f56540d1b68 CR3: 00000001637a8005 CR4: 0000000000772ef0
[ 1815.973542] PKRU: 55555554
[ 1815.973543] Call Trace:
[ 1815.973545] <IRQ> ← 热中断 IRQ 上下文，抢 timer 锁
[ 1815.973548] do_raw_spin_lock+0xbe/0xd0
[ 1815.973553] _raw_spin_lock_irqsave+0x59/0x70
[ 1815.973556] ? lock_timer_base+0x7b/0xa0
[ 1815.973559] lock_timer_base+0x7b/0xa0
[ 1815.973562] add_timer_on+0x70/0x140
[ 1815.973565] __queue_delayed_work+0xac/0x110
[ 1815.973568] queue_delayed_work_on+0x94/0xb0
[ 1815.973571] therm_throt_process+0x164/0x180
[ 1815.973575] intel_thermal_interrupt+0x12b/0x350
[ 1815.973578] __sysvec_thermal+0x46/0x180
[ 1815.973581] sysvec_thermal+0x9f/0xc0
[ 1815.973584] </IRQ>
[ 1815.973585] <TASK> ← 被抢占任务（持锁方）
[ 1815.973589] RIP: 0010:_raw_spin_unlock+0x0/0x50
[ 1815.973601] __get_next_timer_interrupt+0x132/0x380
[ 1815.973605] timer_base_try_to_set_idle+0x17/0x30
[ 1815.973607] tick_nohz_stop_tick+0x70/0x370
[ 1815.973611] tick_nohz_idle_stop_tick+0x72/0xd0
[ 1815.973614] do_idle+0xb8/0x150
[ 1815.973617] cpu_startup_entry+0x30/0x40
[ 1815.974510] NMI backtrace for cpu 5 (sshd：TCP 写路径也在同一把锁上)
[ 1815.974538] do_raw_spin_lock -> lock_timer_base -> __mod_timer -> sk_reset_timer
```

来源：现场 dmesg / trace-cmd NMI backtrace · MetaOS 2.0 · Linux 6.12.90 · SP8000-C340

根因分析：热中断为什么破坏了实时性

高频 Thermal 中断 × tickless 停 tick × timer 锁 —— 硬中断上下文自旋，拖死 RCU

CPU 过热
长时间高负载
触发 Thermal 向量

→

热中断上报
sysvec_thermal
高频进入 Linux 域

→

热节流处理
therm_throt_process
入队延迟工作

→

timer 锁自旋
add_timer_on 抢锁
tickless 停 tick 路径

→

RCU Stall
CPU 10 卡死
Kernel panic

热中断侧(IRQ)：sysvec_thermal → therm_throt_process → add_timer_on → lock_timer_base → 自旋等锁
持锁现场(TASK)：do_idle → tick_nohz_idle_stop_tick → __get_next_timer_interrupt（现场停在 _raw_spin_unlock，临界区未走完）
stalled CPU（4/5、另有 CPU10）均不在 nohz_full=0,1 掩码内

- **持锁方调用流程（拿到锁的完整路径）：**

do_idle → cpuidle_idle_call → tick_nohz_idle_stop_tick → tick_nohz_next_event → get_next_timer_interrupt → __get_next_timer_interrupt
→ raw_spin_lock(&base_local->lock) // base_local = this_cpu_ptr(&timer_bases[BASE_LOCAL])

- 锁冲突：热中断侧 lock_timer_base 抢的正是这把锁 → 硬中断上下文自旋，CPU 钉死、RCU Stall
- 根本原因 · 向量归类错误：THERMAL_APIC_VECTOR 被当作 NMI 式直接分发（非 PIPELINED），绕开管道、无视 inband 屏蔽，强插临界区
- 为什么加了 Dovetail 才死锁：Dovetail 为让 oob 实时任务能抢占 inband 临界区，改用管道 stage 屏蔽替代硬件关中断保护(local_irq_enable/disable)——普通向量被延迟递送、临界区仍安全；热中断（非 PIPELINED）绕开管道屏蔽，成为唯一能强插临界区的向量

实测：cat /proc/interrupts | grep -i TRM（12 核全部命中，累计 7 万+）

```
fiveages@W2-0026: $ cat /proc/interrupts | grep -i TRM
 TRM:       9492      14273      10634      11684       3082       3082       3082       3082       3082       3082       3082       3082   Thermal
```

来源：现场 NMI backtrace / trace-cmd 日志 · /proc/interrupts TRM 实测 · MetaOS 2.0（Linux 6.12 · Xenomai 3.3.1）

解决方案：补丁改了什么、为什么这么改

x86/dovetail：pipeline the thermal APIC sysvec —— 让热中断回归 in-band 管道

补丁思路

- 问题定位：sysvec_thermal 以非管道向量注册（带真实 __apic_eoi()），热负载下与 in-band timer 路径争锁导致 stall。
- 补丁动作：将 THERMAL_APIC_VECTOR 改为管道向量，统一由 do_sysvec_inband() 分发，与 LOCAL_TIMER_VECTOR 同一套机制。
- 安全边界：中断语义与 EOI 行为不变，仅调整分发路径，热中断在 in-band stage 上执行。
- 影响面：3 个文件、+9 / −3 行，只动 x86 热向量分发，其它平台与向量不受影响。

```
/* PIPELINED 宏：补丁后中断先进入 arch_pipeline_entry() */
#define DEFINE_IDTENTRY_IRQ_PIPELINED(func) \
__visible noinstr void func(struct pt_regs *regs, unsigned long error_code) {
arch_pipeline_entry(regs, (u8)error_code); // 先进入管道
} // 热中断回归管道调度，与 LOCAL_TIMER_VECTOR 同机制
```

补丁后的调用路径 —— irq_pipeline.c 注释（提炼）

```
arch_handle_irq(irq)
-> generic_pipeline_irq_desc(desc)
-> handle_apic_irq -> handle_oob_irq // oob 域优先
-> arch_do_IRQ_pipelined -> do_sysvec_inband
-> __sysvec_thermal(regs) // 补丁新增，直接调用
非 PIPELINED 向量（如 NMI）仍直接从 IDT 分发，不进 in-band
```

关键改动

arch/x86/include/asm/idtentry.h

DECLARE_IDTENTRY_SYSVEC（非管道）
→ DECLARE_IDTENTRY_SYSVEC_PIPELINED

arch/x86/kernel/irq.c

DEFINE_IDTENTRY_SYSVEC_PIPELINED 定义
EOI：__apic_eoi() → apic_eoi()

arch/x86/kernel/irq_pipeline.c

do_sysvec_inband() 新增 THERMAL_APIC_VECTOR 分发
run_sysvec_on_irqstack_cond(__sysvec_thermal)

一句话总结

热中断回归 in-band 管道，语义与 EOI 不变。验证：8/8–8/9 家用 → 8/10–8/12 客户现场 → 8/14–8/16 温度复测，均未复现；v1 已提交社区评审。

来源：x86/dovetail · [PATCH v1] pipeline the thermal APIC sysvec · 3 files changed · +9 / −3

从发现到合入：一个社区补丁的生命周期

问题修完不是终点——把它变成社区主线代码，才是对所有用户的价值。

01

起草与自测

写清楚问题、复现方法与修复思路，多平台回归验证。

02

邮件列表提交

发送到 Xenomai 邮件列表，附上完整背景与测试数据。

03

维护者评审

回应维护者意见，多轮修订，覆盖边界情况与兼容性。

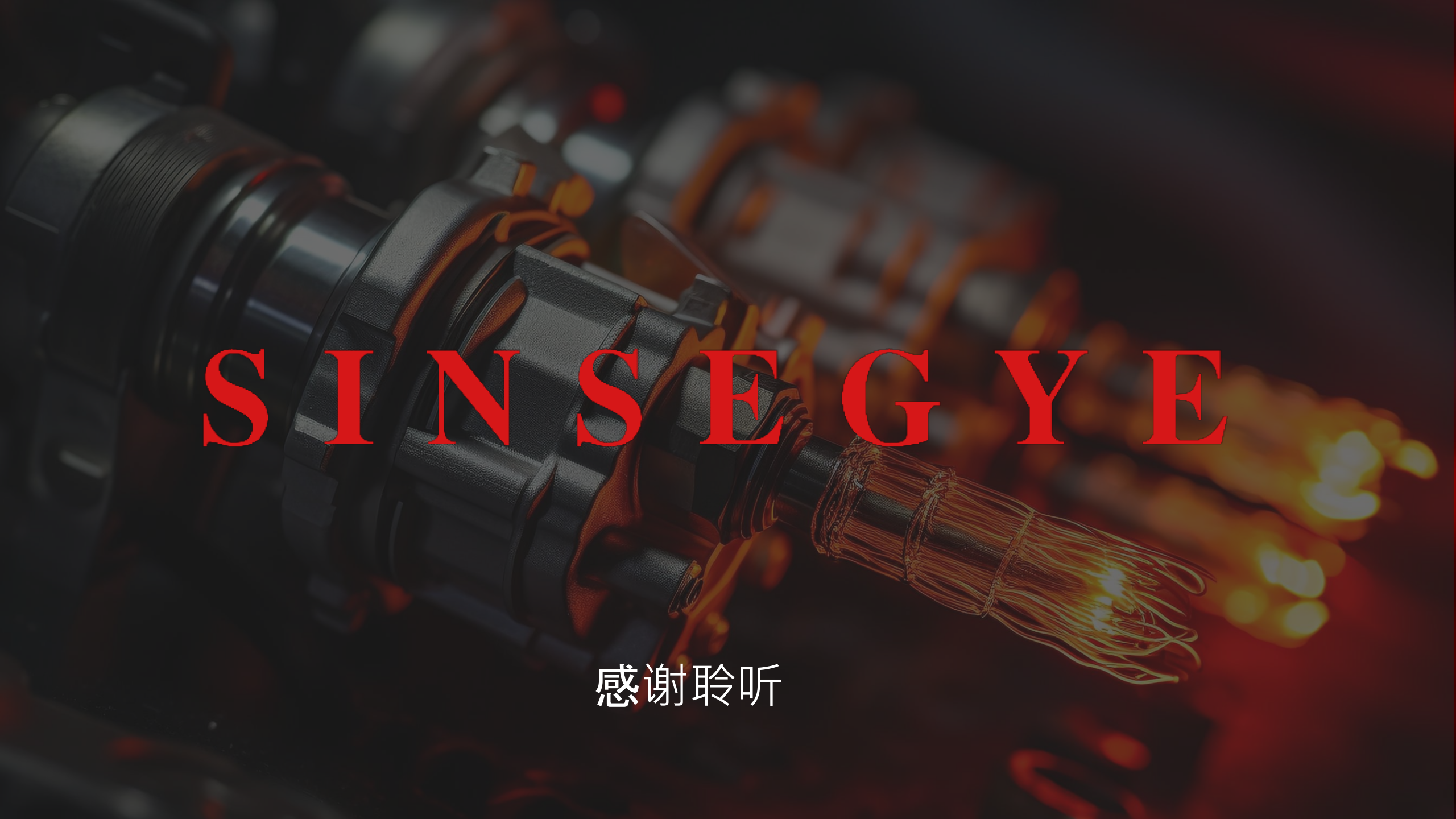
04

合入主线

进入下一版本，让所有实时 / 机器人用户受益。

· 提交补丁不只是"代码能跑"，更是向社区说清楚：这是个真问题、有数据、修法安全。
· 对我们而言，这也是反哺上游、把机器人一线遇到的实时性问题反馈给 Xenomai 的过程。
• **补丁稳定性验证：**stress -c 14 -d 14 -i 14 -m 16 压 CPU/磁盘/内存，glmark2 压核显，gpu_burn -d 360000 压 NVIDIA RTX 4060 独显，连续 7×24 小时未再出现卡死。

刘杨 688181



感谢聆听

Open Source 西门子

Beijing · 2026

Oct. 29
Industrial Open Source

Oct. 30
Open Source Robotics

Agenda