openEuler 24.03 LTS SP3 技术白皮书

1. 概述

OpenAtom openEuler（简称“openEuler”） 社区是一个面向数字基础设施操作系统的开源社区。由开放原子开源基金会（以下简称“基金会”）孵化及运营。

openEuler 是一个面向数字基础设施的操作系统，支持服务器、云计算、边缘计算、嵌入式等应用场景，支持多样性计算，致力于提供安全、稳定、易用的操作系统。通过为应用提供确定性保障能力，支持 OT 领域应用及 OT 与 ICT 的融合。

openEuler 社区通过开放的社区形式与全球的开发者共同构建一个开放、多元和架构包容的软件生态体系，孵化支持多种处理器架构、覆盖数字基础设施全场景，推动企业数字基础设施软硬件、应用生态繁荣发展。

2019 年 12 月 31 日，面向多样性计算的操作系统开源社区 openEuler 正式成立。

2020 年 3 月 30 日， openEuler 20.03 LTS （Long Term Support，简写为 LTS，中文为长生命周期支持）版本正式发布，为 Linux 世界带来一个全新的具备独立技术演进能力的 Linux 发行版。

2020 年 9 月 30 日，首个 openEuler 20.09 创新版发布，该版本是 openEuler 社区中的多个企业、团队、独立开发者协同开发的成果，在 openEuler 社区的发展进程中具有里程碑式的意义，也是中国开源历史上的标志性事件。

2021 年 3 月 31 日，发布 openEuler 21.03 内核创新版，该版本将内核升级到 5.10，并在内核方向实现内核热升级、内存分级扩展等多个创新特性，加速提升多核性能，构筑千核运算能力。

2021 年 9 月 30 日，全新 openEuler 21.09 创新版如期而至，这是 openEuler 全新发布后的第一个社区版本，实现了全场景支持。增强服务器和云计算的特性，发布面向云原生的业务混部 CPU 调度算法、容器化操作系统 KubeOS 等关键技术；同时发布边缘和嵌入式版本。

2022 年 3 月 30 日，基于统一的 5.10 内核，发布面向服务器、云计算、边缘计算、嵌入式的全场景 openEuler 22.03 LTS 版本，聚焦算力释放，持续提升资源利用率，打造全场景协同的数字基础设施操作系统。

2022 年 9 月 30 日，发布 openEuler 22.09 创新版本，持续补齐全场景的支持。

2022 年 12 月 30 日，发布 openEuler 22.03 LTS SP1 版本，打造最佳迁移工具实现业务无感迁移，性能持续领先。

2023 年 3 月 30 日，发布 openEuler 23.03 内核创新版本，采用 Linux Kernel 6.1 内核，为未来 openEuler 长生命周期版本采用 6.x 内核提前进行技术探索，方便开发者进行硬件适配、基础技术创新及上层应用创新。

2023 年 6 月 30 日，发布 openEuler 22.03 LTS SP2 版本，场景化竞争力特性增强，性能持续提升。

2023 年 9 月 30 日，发布 openEuler 23.09 创新版本，是基于 6.4 内核的创新版本（参见版本生命周期），提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2023 年 11 月 30 日，发布 openEuler 20.03 LTS SP4 版本，其作为 20.03 LTS 版本的增强扩展版本，面向服务器、云原生、边缘计算场景，提供更多新特性和功能增强。

2023 年 12 月 30 日，发布 openEuler 22.03 LTS SP3 版本，是 22.03 LTS 版本增强扩展版本，面向服务器、云原生、边缘计算和嵌入式场景，持续提供更多新特性和功能扩展，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2024 年 5 月 30 日，发布 openEuler 24.03 LTS，基于 6.6 内核的长周期 LTS 版本（参见版本生命周期），面向服务器、云、边缘计算、AI 和嵌入式场景，提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2024 年 6 月 30 日，发布 openEuler 22.03 LTS SP4，是 22.03 LTS 版本增强扩展版本，面向服务器、云原生、边缘计算和嵌入式场景，持续提供更多新特性和功能扩展，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2024 年 9 月 30 日，发布 openEuler 24.09，基于 6.6 内核的创新版本，提供更多新特性和功能。

2024 年 12 月 30 日，发布 openEuler 24.03 LTS SP1，基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、边缘计算和嵌入式场景，持续提供更多新特性和功能扩展，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

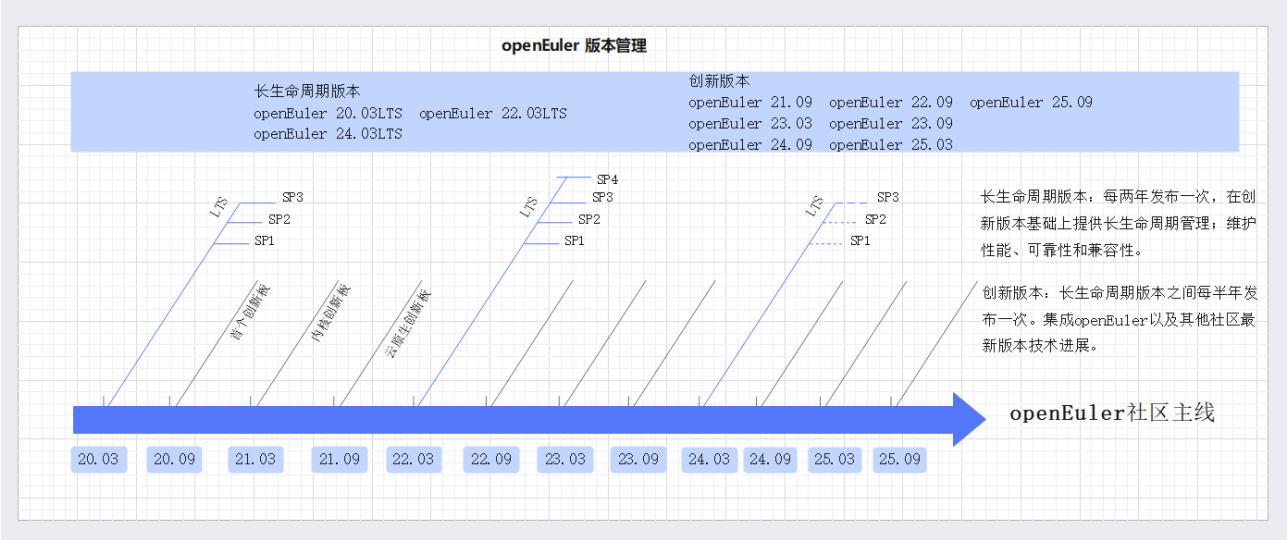
2025 年 3 月 30 日，发布 openEuler 25.03 是基于 6.6 内核的创新版本，面向服务器、云、边缘计算和嵌入式场景，提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2025 年 6 月 30 日，发布 openEuler 24.03 LTS SP2，基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、AI 和嵌入式场景，持续提供更多新特性和功能扩展，包括内核优化、异构协同推理、众核高密、机密容器、多核多实例混部等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2025 年 9 月 30 日，发布 openEuler 25.09，基于 6.6 内核创新版本，面向服务器、云、AI 和嵌入式场景，持续提供更多新特性和功能扩展，包括内核优化、异构协同推理、众核高密、机密容器、机密虚机、

多核多实例混部、编译器、可信计算、密码加速等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

2025 年 12 月 30 日，openEuler 首个支持超节点的版本正式发布。新版本 openEuler 24.03 LTS SP3 是基于 6.6 内核的 24.03-LTS 版本增强扩展版本（参见版本生命周期），面向服务器、云、AI 场景，持续提供更多新特性和功能扩展，包括内核优化、异构协同推理、智能诊断、机密虚机、编译器、RISC-V 架构优化、智能开发者桌面、安全加固、灵衢超节点、身份认证、虚拟化等，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

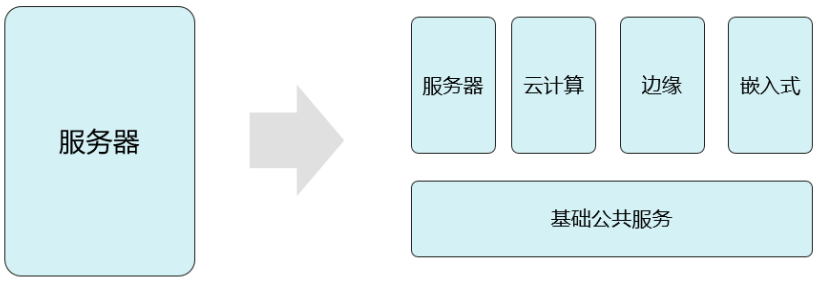


openEuler 作为一个操作系统发行版平台，其 LTS 版本为企业级用户提供一个安全稳定可靠的操作系统。

openEuler 也是一个技术孵化器。通过每半年发布一个创新版，快速集成 openEuler 以及其他社区的最新技术成果，将社区验证成熟的特性逐步回合到发行版中。这些新特性以单个开源项目的方式存在于社区，方便开发者获得源代码，也方便其他开源社区使用。

社区中的最新技术成果持续合入社区发行版，社区发行版通过用户反馈反哺技术，激发社区创新活力，从而不断孵化新技术。发行版平台和技术孵化器互相促进、互相推动、牵引版本持续演进。

openEuler 覆盖全场景的创新平台



openEuler 已支持 X86、ARM、RISC-V、LoongArch 多处理器架构，持续完善多样性算力生态体验。

openEuler 社区面向场景化的 SIG 不断组建，推动 openEuler 应用边界从最初的服务器场景，逐步拓展到云计算、边缘计算、嵌入式等更多场景，openEuler 正成为覆盖数字基础设施全场景的操作系统。

openEuler 希望与广大生态伙伴、用户、开发者一起，通过联合创新、社区共建，不断增强场景化能力，最终实现统一操作系统支持多设备，应用一次开发覆盖全场景。

openEuler 开放透明的开源软件供应链管理

开源操作系统的构建过程，也是供应链聚合优化的过程。拥有可靠开源软件供应链，是大规模商用操作系统的基础。 openEuler 从用户场景出发，回溯梳理相应的软件依赖关系，理清所有软件包的上游社区地址、源码和上游对应验证。完成构建验证、分发、实现生命周期管理。开源软件的构建、运行依赖关系、上游社区，三者之前形成闭环且完整透明的软件供应链管理。

2. 平台架构

系统框架

openEuler 是覆盖全场景的创新平台，在引领内核创新，夯实云化基座的基础上，面向计算架构互联总线、存储介质发展新趋势，创新分布式、实时加速引擎和基础服务，结合边缘、嵌入式领域竞争力探索，打造全场景协同的面向数字基础设施的开源操作系统。

openEuler 24.03 LTS SP3 发布面向服务器、云原生、边缘和嵌入式场景的全场景操作系统版本，统一基于 Linux Kernel 6.6 构建，对外接口遵循 POSIX 标准，具备天然协同基础。

同时 openEuler 24.03 LTS SP3 版本集成分布式软总线、K3s 边云协同框架等能力，进一步提升数字基础设施协同能力，构建万物互联的基础。

面向未来，社区将持续创新、社区共建、繁荣生态，夯实数字基座。

夯实云化基座

容器操作系统 KubeOS：云原生场景，实现 OS 容器化部署、运维，提供与业务容器一致的基于 K8S 的管理体验。

安全容器方案：iSulad+shimv2+StratoVirt 安全容器方案，相比传统 Docker+QEMU 方案，底噪和启动时间优化 40%。

机密容器方案：iSulad+Kuasar+secGear 机密容器方案，提供兼容云原生生态的隐私数据保护方案。

机密计算方案：基于 ARM CCA 机密计算架构规范，首个支持 CCA 机密虚机的社区版本，提供机密虚机内数据和代码的机密性和完整性保护，即使面对拥有特权的基础设施软件或云服务提供商，也能得到有效保护。

全场景

边缘计算：发布面向边缘计算场景的版本，支持 K3s 边云协同框架，具备边云应用统一管理和发放等基础能力。

嵌入式：发布面向嵌入式领域的版本，该版本镜像大小 < 5M，启动时间 < 5s；弹性虚拟化底座以及混合关键性部署框架，支持多核多实例同时部署，跨 OS 通信为不同 OS 之间提供一套基于共享内存的高效通信机制。

AI 原生 OS：OS 使能 AI 软件栈，开箱即用；异构融合内存，调度，训推场景降本增效；智能化交互平台，赋能开发者及管理员。

超节点 OS：OS 基于原有框架扩展支持超节点，分层构建超节点系统软件栈，提供超节点能力和接口，赋能超节点生态落地和价值释放。

繁荣社区生态

友好桌面环境：UKUI、DDE 、Kiran-desktop、GNOME 桌面环境，丰富社区桌面环境生态。

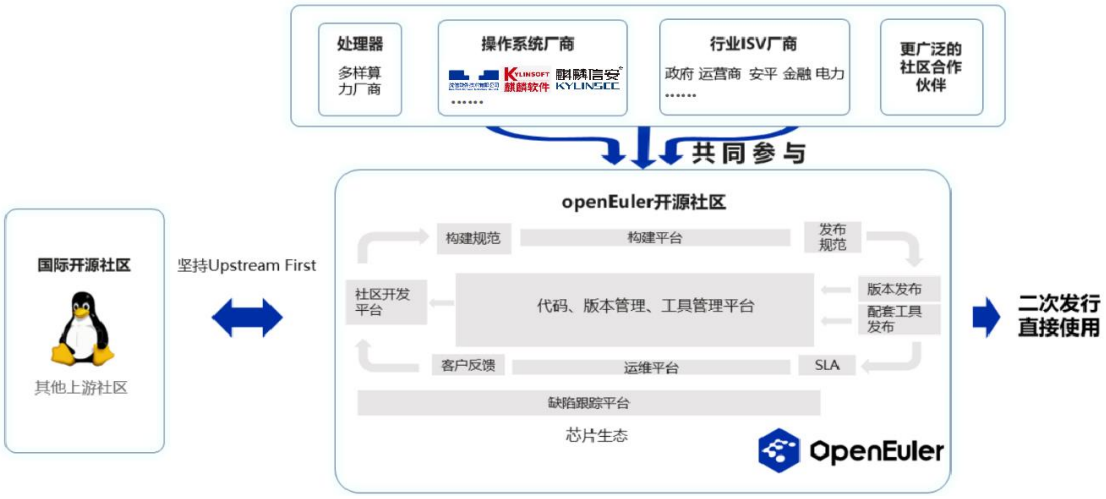
openEuler DevKit：支持操作系统迁移、兼容性评估、简化安全配置 secPaver 等更多开发工具。

DevStation 开发者工作站：基于 openEuler 的智能开发者工作站，旨在提供开箱即用、高效安全的开发环境，打通从部署、编码、编译、构建到发布的全流程，通过新增 MCP AI

智能引擎，快速完成社区工具链调用，实现从基础设施搭建到应用开发的效率飞跃，深度集成 CVE 智能修复系统。极大提升了安全漏洞的响应与修复效率。

平台框架

openEuler 社区与上下游生态建立连接，构建多样性的社区合作伙伴和协作模式，共同推进版本演进。



硬件支持

openEuler 社区当前已与多个设备厂商建立丰富的南向生态，比如 Intel、AMD 等主流芯片厂商的加入和参与，openEuler 全版本支持 x86、Arm、龙芯、RISC-V 五种架构，并支持多款 CPU 芯片，包括龙芯 3 号 、兆芯开先/开胜系列、Intel Sierra Forest/Granite Rapids、AMD EPYC 4/5 等芯片系列，支持多个硬件厂商发布的多款整机型号、板卡型号，支持网卡、RAID、FC、GPU&AI、DPU、SSD、安全卡七种类型的板卡，具备良好的兼容性。

支持的 CPU 架构如下：

硬件类型	x86_64	arm64	LoongArch	RISC-V
CPU	Intel、AMD、Hygon、兆芯	鲲鹏、飞腾	龙芯	Sophgo、THead 等

全版本支持的硬件型号可在兼容性网站查询兼容性列表。

3. 运行环境

服务器

若需要在物理机环境上安装 openEuler 操作系统，则物理机硬件需要满足以下兼容性和最小硬件要求。

硬件兼容支持请查看 openEuler 兼容性列表：兼容性列表。

部件名称	最小硬件要求
架构	ARM64、x86_64、riscV、LoongArch64
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

虚拟机

openEuler 安装时，应注意虚拟机的兼容性问题，当前已测试可以兼容的虚拟机及组件如下所示。

1. 以 openEuler 24.03 LTS SP3 为 HostOS，组件版本如下：

- libvirt-9.10.0-24.oe2403sp3
- libvirt-client-9.10.0-24.oe2403sp3
- libvirt-daemon-9.10.0-24.oe2403sp3
- qemu-8.2.0-57.oe2403sp3
- qemu-img-8.2.0-57.oe2403sp3

2. 兼容的虚拟机列表如下：

HostOS	GuestOS(虚拟机)	架构
openEuler 24.03 LTS SP3	Centos 6	x86_64
openEuler 24.03 LTS SP3	Centos 7	aarch64
openEuler 24.03 LTS SP3	Centos 7	x86_64
openEuler 24.03 LTS SP3	Centos 8	aarch64
openEuler 24.03 LTS SP3	Centos 8	x86_64
openEuler 24.03 LTS SP3	Windows Server 2016	x86_64
openEuler 24.03 LTS SP3	Windows Server 2019	x86_64

部件名称	最小虚拟化空间要求
架构	ARM64、x86_64
CPU	2 个 CPU
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

边缘设备

若需要在边缘设备环境上安装 openEuler 操作系统，则边缘设备硬件需要满足以下兼容性和最小硬件要求。

部件名称	最小硬件要求
架构	ARM64、x86_64
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

嵌入式

若需要在嵌入式环境上安装 openEuler Embedded 操作系统，则嵌入式硬件需要满足以下兼容性和最小硬件要求。

部件名称	最小硬件要求
架构	ARM64、ARM32、x86_64
内存	为了获得更好的体验，建议不小于 512MB
硬盘	为了获得更好的体验，建议不小于 256MB

4. 场景创新

AI 场景创新

智能时代，操作系统需要面向 AI 不断演进。一方面，在操作系统开发、部署、运维全流程以 AI 加持，让操作系统更智能；另一方面，openEuler 已支持 ARM，x86，RISC-V 等全部主流通用计算架构，在智能时代，openEuler 也率先支持 NVIDIA、昇腾等主流 AI 处理器，成为使能多样性算力的首选。

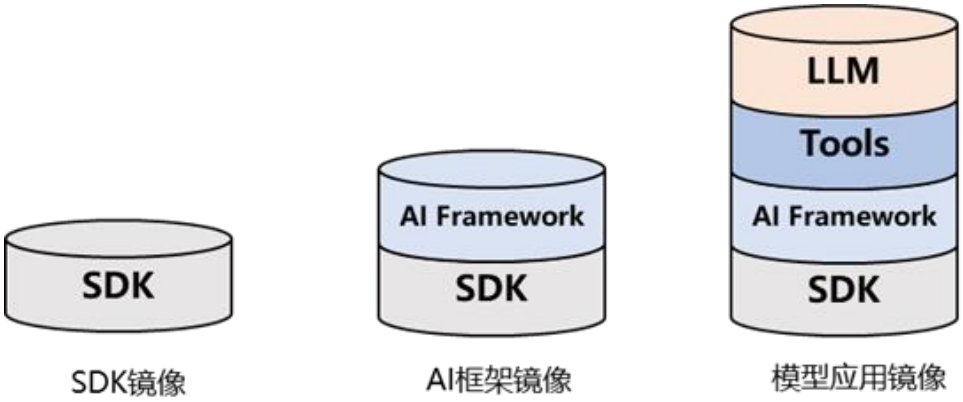
OS for AI

开箱易用

openEuler 兼容 NVIDIA、Ascend 等主流算力平台的软件栈，为用户提供高效的开发运行环境。通过将不同 AI 算力平台的软件栈进行容器化封装，即可简化用户部署过程，提供开箱即用的体验。同时，openEuler 也提供丰富的 AI 框架，方便大家快速在 openEuler 上使用 AI 能力。

功能描述

1. openEuler 已兼容 CANN、CUDA 等硬件 SDK，以及 TensorFlow、PyTorch、MindSpore 等相应的 AI 框架软件，支持 AI 应用在 openEuler 上高效开发与运行。
2. openEuler AI 软件栈容器化封装优化环境部署过程，并面向不同场景提供以下三类容器镜像。



- SDK 镜像：以 openEuler 为基础镜像，安装相应硬件平台的 SDK，如 Ascend 平台的 CANN 或 NVIDIA 的 CUDA 软件。
- AI 框架镜像：以 SDK 镜像为基础，安装 AI 框架软件，如 PyTorch 或 TensorFlow。此外，通过此部分镜像也可快速搭建 AI 分布式场景，如 Ray 等 AI 分布式框架。
- 模型应用镜像：在 AI 框架镜像的基础上，包含完整的工具链和模型应用。

相关使用方式请参考 openEuler AI 容器镜像用户指南。

应用场景

openEuler 使能 AI，向用户提供更多 OS 选择。基于 openEuler 的 AI 容器镜像可以解决开发运行环境部署门槛高的问题，用户根据自身需求选择对应的容器镜像即可一键部署，三类容器镜像的应用场景如下。

- SDK 镜像：提供对应硬件的计算加速工具包和开发环境，用户可进行 Ascend CANN 或 NVIDIA CUDA 等应用的开发和调试。同时，可在该类容器中运行高性能计算任务，例如大规模数据处理、并行计算等。
- AI 框架镜像：用户可直接在该类容器中进行 AI 模型开发、训练及推理等任务。
- 模型应用镜像：已预置完整的 AI 软件栈和特定的模型，用户可根据自身需求选择相应的模型应用镜像来开展模型推理或微调任务。

sysHAX 大语言模型异构协同加速运行时

功能描述

sysHAX 大语言模型异构协同加速运行时专注于单机多卡环境下大模型推理任务的性能提升，针对鲲鹏+xPU（GPU、NPU 等）的异构算力协同，显著提升大模型的吞吐量和并发量：

- CPU 推理加速：通过 NUMA 亲和调度、矩阵运算并行加速、SVE 指令集推理算子适配等方式，提升 CPU 的吞吐量。
- 异构融合调度：支持在 GPU 侧任务满载时，动态将推理请求的 prefill 阶段在 GPU 上执行，decode 阶段放在 CPU 上执行。

应用场景

sysHAX 大语言模型推理优化方案当前支持 DeepSeek、Qwen 等 transformer 架构的模型。其中，CPU 推理加速能力已完成了对 DeepSeek 7B、14B、32B 以及 Qwen2.5、Qwen3 系列模型的适配。目前支持的硬件包括 Nvidia、昇腾 310P 等主流硬件。sysHAX 主要适用于以下典型场景：

- 数据中心场景：sysHAX 通过上述技术，利用 CPU 填充推理任务，充分利用 CPU 资源，增加大模型并发量与吞吐量。

sysHAX 使用方式请参考 sysHAX 部署指南。

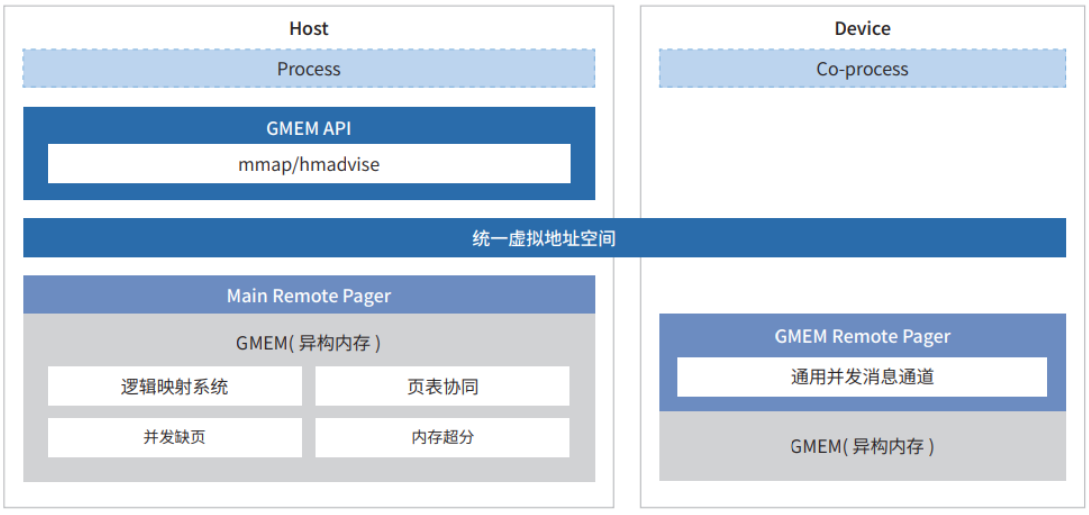
异构融合 GMem

在后摩尔时代，GPU、TPU 和 FPGA 等专用异构加速器设备正不断涌现，它们与 CPU 类似，需要将数据放在本地内存（例如 LPDDR 或 HBM）中以提高计算速度。加速器厂商们也不可避免地需要开发复杂的内存管理系统。 现行加速器内存管理方案存在诸多缺陷：

- CPU 侧内存管理与加速器侧分离，数据显式搬移，加速器内存管理的易用性和性能难以平衡。
- 大模型场景下加速器设备 HBM 内存（High BandWidth Memory）严重不足，现有的手动 swap 方案性能损耗大且 通用性差。
- 搜推、大数据场景存在大量无效数据搬移，缺少高效内存池化方案。

Linux 现有的 HMM 框架，编程复杂度高且依赖人工调优，性能和可移植性差，引发 OS 社区反弹，最终导致 HMM 方案搁浅。异构加速器领域亟需高效的统一内存管理机制。异构通用内存管理框架 GMem （Generalized Memory Management），提供了异构内存互联的中心化管理机制，且 GMem API 与 Linux 原生内存管理 API 保持统一，易用性强，性能与可移植性好。加速器使用 GMem API 将内存接入统一地址空间后，可自动获得 GMem 面向异构内存编程优化的能力。与此同时，加速器驱动无需重复实现内存管理框架，大幅降低开发维护带来的成本。开发者使用一套统一申请、释放的 API，即可完成异构内存编程，无需处理内存搬移等细节。在加速器 HBM 内存不足时，GMem 可将 CPU 内存作为加速器缓存，透明地超分 HBM，无需应用手动 swap。GMem 提供高效免搬移的内存池化方案，当内存池以共享方式接入后，可解决数据反复搬移的痛点。

功能描述



GMem 革新了 Linux 内核中的内存管理架构，其中逻辑映射系统屏蔽了 CPU 和加速器地址访问差异，remote_pager 内存消息交互框架提供了设备接入抽象层。在统一的地址空间下，GMem 可以在数据需要被访问或换页时，自动地迁移数据到 OS 或加速器端。

- **异构内存特性**

为了结合加速器算力与 CPU 通用算力，实现统一的内存管理和透明内存访问，GMem 设计了统一虚拟内存地址空间机制，将原本的 OS 与加速器并行的两套地址空间合并为统一虚拟地址空间。

GMem 建立了一套新的逻辑页表去维护这个统一虚拟地址空间，通过利用逻辑页表的信息，可以维护不同处理器、不同微架构间多份页表的一致性。基于逻辑页表的访存一致性机制，内存访问时，通过内核缺页流程即可将待访问内存在主机与加速器进行搬移。在实际使用时，加速器可在内存不足时可以借用主机内存，同时回收加速器内的冷内存，达到内存超分的效果，突破模型参数受限于加速器内存的限制，实现低成本的大模型训练。

- **分级内存管理**

通过在内核中提供 GMem 高层 API，允许加速器驱动通过注册 GMem 规范所定义的 MMU 函数直接获取内存管理功能， 建立逻辑页表并进行内存超分。逻辑页表将内存管理的高层逻辑与 CPU 的硬件相关层解耦，从而抽象出能让各类加速器复用的高层内存管理逻辑。加速器只需要注册底层函数，不再需要实现任何统一地址空间协同的高层逻辑。 由系统自动管理数据在不同存储介质间的迁移，例如 dram 和 hbm。

- **remote Pager 内存消息交互框架**

Remote Pager 作为 OS 内核外延的内存管理框架，设计并实现了主机和加速器设备之间协作的消息通道、进程管理、 内存交换和内存预取等模块，由独立驱动 remote_pager.ko 使能。通过 Remote Pager 抽象层可以让第三方加速器很容易地接入 GMEM 系统，简化设备适配难度。

- **用户 API**

用户可以直接使用 OS 的 mmap 分配统一虚拟内存，GMem 在 mmap 系统调用中新增分配统一虚拟内存的标志（MMAP_ PEER_SHARED）。

同时 libgmem 用户态库提供了内存预取语义 hmadvise 接口，协助用户优化加速器内存访问效率（参考 https://atomgit.com/openeuler/libgmem）。

- **约束限制**

硬件约束：需要使用昇腾 npu 卡。

软件约束：目前仅支持 2M 大页，所以 host OS 以及 NPU 卡内 OS 的透明大页需要默认开启。 通过 MAP_PEER_SHARED 申请的异构内存目前不支持 fork 时继承。

GMem 使用方法可参考以下链接：

https://atomgit.com/openeuler/docs-centralized/tree/master/docs/zh/docs/GMEM

应用场景

- 异构统一内存编程

在面向异构内存编程时，使用 GMEM 可分配 CPU 和加速器之间的统一虚拟内存，CPU 内存与加速器内存可共享一个指针，显著降低了异构编程复杂度。当前基于 NPU 试点，驱动仅需百行修改即可接入 GMEM，替换原有约 4000 行内存管理框架代码。

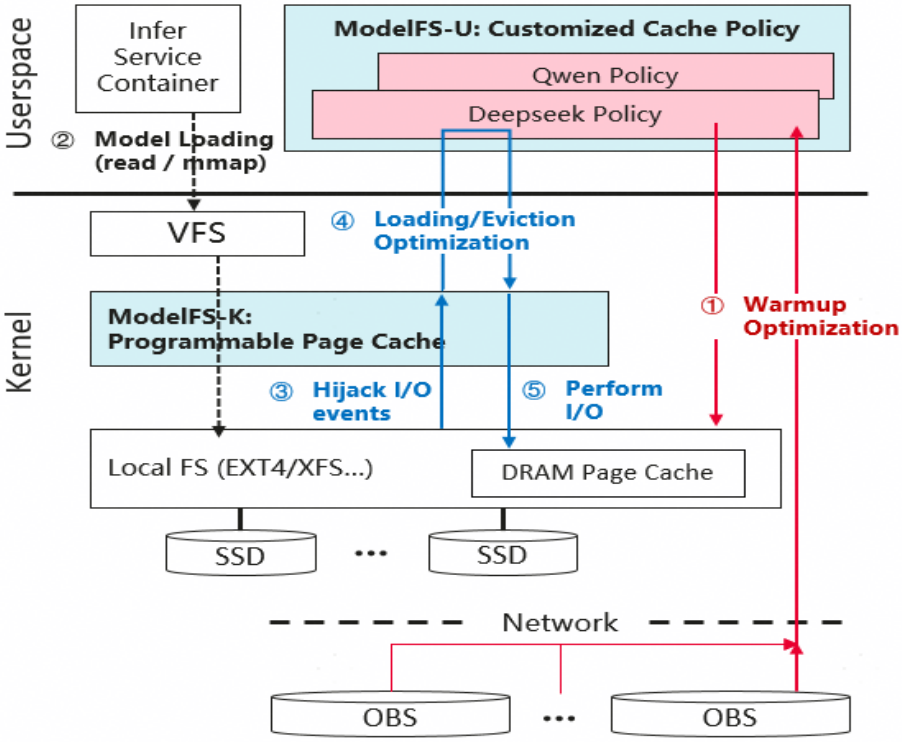
- 加速器内存自动超分

使用 GMEM 接口分配内存时，将不受加速器的物理内存容量所限制，应用可以透明地超分内存（当前上限为 CPU 的 DRAM 容量）。GMEM 将较冷的设备内存页换出到 CPU 内存上，拓展了应用处理的问题规模，实现高性能、低门槛训推。 通过 GMEM 提供的极简异构内存管理框架，在超大模型训练中，GMEM 性能领先 NVIDIA-UVM。随着内存使用量增长，领先比例不断提升，在超分两倍以上时可领先 NVIDIA-UVM 60% 以上（数据基于 NPU-Ascend910 与 GPU-A100 硬件，在 相同 HBM 内存条件下测试）。

ModelFS 模型启动加速

ModelFS 针对大语言模型（LLM）推理启动阶段的模型加载瓶颈进行优化。 现有相关工作主要通过优化推理框架来提升模型加载性能，这种方法往往以牺牲兼容性为代价。然而，基于我们的工业实践经验，兼容性是决定一项技术能否在实际场景中广泛应用的关键因素。本工作在保证强兼容性的前提下，通过优化文件系统的缓存策略，实现了模型加载性能的领先水平。 ModelFS 在内核中设计了一个非侵入式、灵活且轻量级的可编程页缓存框架，允许用户自定义文件系统的页缓存策略。 基于可编程页缓存，我们进一步设计了面向模型加载优化的缓存策略的参考实现。

功能描述



ModelFS-K：

ModelFS 的内核模块，提供文件系统（FS）页缓存可编程框架。核心设计是一个堆叠 FS，可以挂载到现有 FS 之上，ModelFS-K 将底层文件系统的相关逻辑用 UPC（用户态调用）重定向到用户态实现的 prefetch 和 evict 函数。

ModelFS-U：

ModelFS 的用户态模块，提供缓存策略的运行时。提供一套 VFS-like 的用户态编程框架，模型实现者/IO 优化者可以根据模型的加载 IO 特性自定义缓存策略。用户需实现 init()，exit()， prefetch()，evict()，ModelFS-U 会将它们注册到 ModelFS-K 中，并在运行过程中负责解析 IO 事件和进行异步数据预取/淘汰。

应用场景

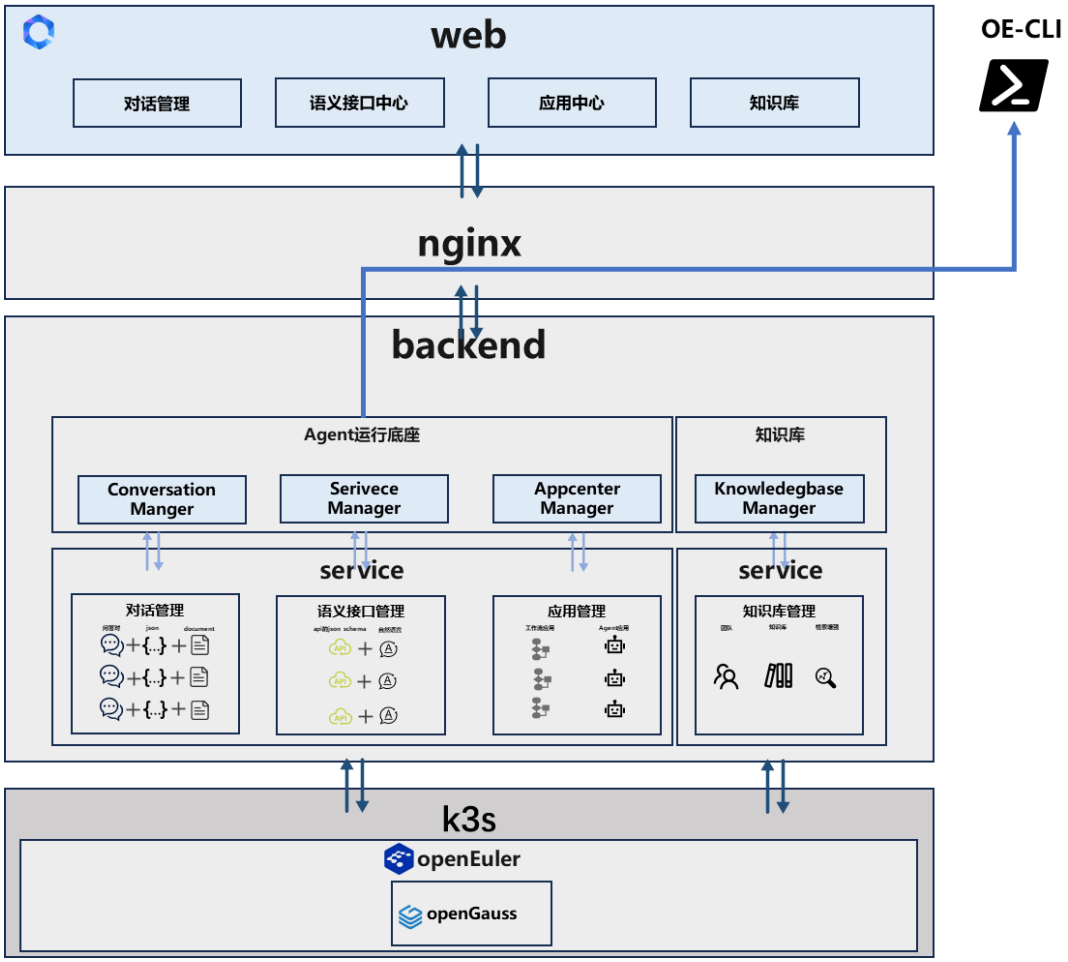
ModelFS 可用于多种类型存储后端的 IO 加速，包括本地 FS，分布式 FS，对象存储（OBS）等，以下是两种典型的用法。

1) 远端 OBS 存储权重预热：对于模型权重存储于远端 OBS 的情况，由于 OBS 带宽低，推理系统往往会将热点权重提前预热到本地存储。ModelFS 可以通过实现 IO 聚合的缓存策略，将多个 OBS 桶的带宽聚合来加速权重预热。

2) 本地 SSD 权重加载加速：对于模型权重存储在本地 SSD 的情况，性能瓶颈在于本地 SSD 带宽利用率低。ModelFS 实现了一种基于 I/O 模板的机制，在第一次运行时追踪每个推理服务的 IO 加载行为，而在后续推理启动过程中可以精确感知未来的 IO 行为，进而充分利用 SSD 带宽、XPU 亲和性以提升预取与淘汰机制的效率。

AI for OS

当前，openEuler 和 AI 深度结合，一方面，基于 openEuler 操作系统，研发出了 openEuler Intelligence，初步实现基于知识库的智能问答、基于语义接口的工作流编排、基于 mcp 的 Agent 构建等功能，在此基础上，openEuler Intelligence 集成了部分系统服务，让 openEuler 更智能。



智能问答

功能描述

openEuler Intelligence 目前支持 Web 和智能 Shell 两个入口。

- Web 入口：用户可以通过 Web 入口以可视化的形式进行知识库的构建和使用、知识库准确率的自动化测试与评估结果获取、openapi 形式的语义接口注册、基于语义接口的工作流应用的构建和使用、mcp 的注册安装和激活和基于 mcp 的 Agent 应用的构建和使用，Web 入口便携了新手用户对 openEuler 知识的获取和对 openEuler AI 能力的使用。
- 智能 Shell 入口：用户可以通过 Shell 入口调用智能体框架中的智能问答或者是预集成的 Agent 应用，Shell 入口便携了运维人员对操作系统的使用，能通过亲和操作系统的方式与 openEuler 进行智能交互，降低运维成本。

智能规划、调度和推荐

- 智能规划：openEuler Intelligence 的 Agent 应用可以基于用户的输入和当前可用的工具实时规划运行步骤，直至完成用户目标或者达到步骤执行上限。
- 智能调度：openEuler Intelligence 支持用户在一个工作流应用中定义多个工作流，基于用户的查询，openEuler Intelligence 会自动的提取参数且选择最为合适的工作流进行工作。
- 智能推荐：openEuler Intelligence 基于用户的查询和工作流的运行结果，推荐用户接下来可能会使用的工作流，增加任务的完成概率，简便应用的使用。

工作流应用

- 语义接口：语义接口是指含有自然语言注释的接口形式，openEuler Intelligence 提供了两种语义接口注册方式：首先，openEuler Intelligence 允许用户将 api 接口以 openapi（3.0+）yaml 文件形式注册到系统中，用户需要在编写 openapi yaml 文件时，对接口添加自然语言注释，后续在这些接口调用过程中，大模型会根据接口的注释对接口进行选择和填参；其次，openEuler Intelligence 也允许用户将功能以 python 编码的形式注册到 openEuler Intelligence 中，这种形式对于 openEuler Intelligence 而言更为亲和。以上两种形式产生的语义接口最终都可以通过可视化形式编排为工作流。
- 工作流编排及调用：openEuler Intelligence 允许用户将系统提供的语义接口以及用户注册的语义接口以可视化的形式连线成工作流，并支持用户对工作流进行调试且以应用的形式进行发布和使用，工作流在调试和使用过程中会展示中间结果，降低用户调试成本，提升用户交互体验。

Agent 应用

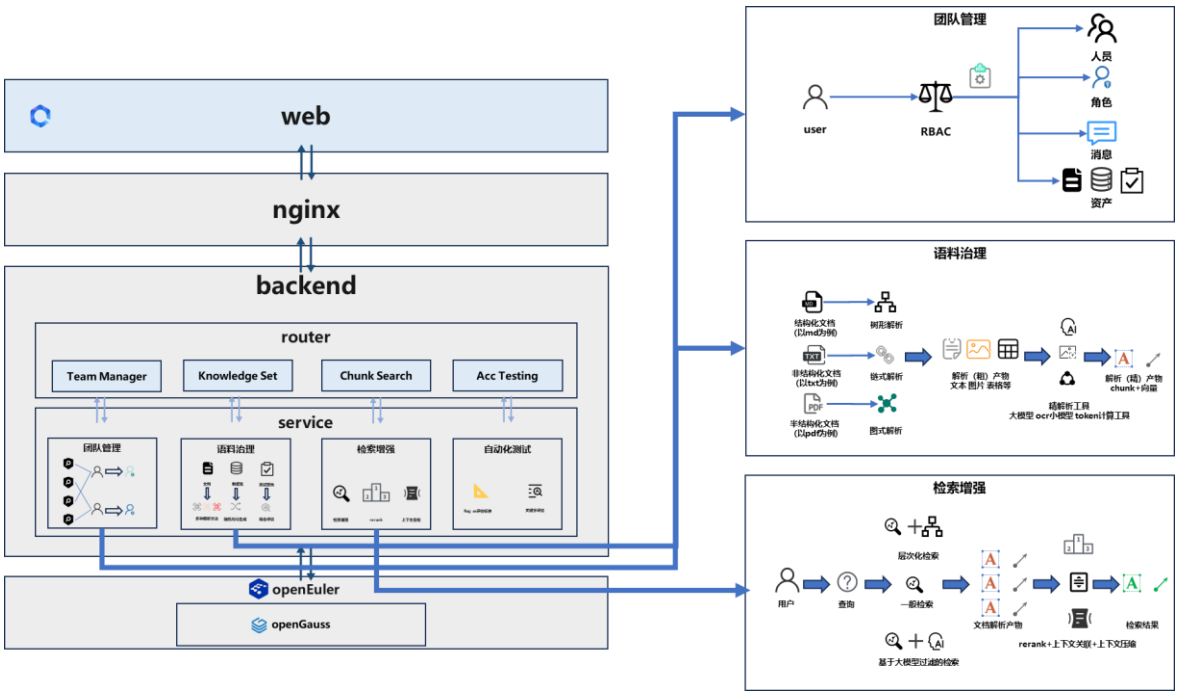
- mcp 注册、安装和激活：mcp 是当前比较主流的一种 AI 相关协议，它支持用 sdk 将复杂多样服务统一封装，带有天然的语义信息，并支持 AI 对基于 mcp 改造后服务下的工具进行比较便捷的调用。
- Agent 构建和应用：当前 intelligence 支持以 mcp 和不同的大模型结合构建 Agent，这些构建完成的 Agent 可以基于配置的大模型信息以及后续用户输入的目标，将用户的目标拆解成阶段性需求，最后使用 mcp 服务下的工具将阶段性需求完成，直至达成用户的目标。

RAG

RAG(检索增强技术)是为了增强大模型长期记忆能力和降低大模型训练成本诞生的技术，相较传统 RAG，openEuler Intelligence 中的 RAG 技术在检索前处理、知识索引、检索增强算法和检索后处理方面做了改进：

- **检索前处理**：对于信息缺失的用户查询，基于历史上下文和用户意图，提供查询改写的能力，提升检索准确率；
- **知识索引**：对于格式和内容多样化的文档，提供摘要、文本特征提取、树形解析、ocr 等文档解析能力，建立片段特征索引，提升检索增强命中率；
- **检索增强算法**：对于多样化的检索场景，提供 8 种检索方法，其中基于动态关键字权重的检索方法和基于大模型过滤的检索方法能极大提升开箱及用的准确率；
- **检索后处理**：对于上下文缺失的片段，提供均匀随机化上下文补全的方法，增加片段信息完整度，降低最终模型拟合幻想程度；对于 token 阈值上限的片段（集合），提供基于杰卡德距离的 rerank 方法、基于通用词去除及随机丢弃的 token 压缩方法和基于二分的 token 截断方法，允许一些资源受限的场景也能正常进行智能问答。

通过以上能力，相较传统的 RAG 技术，openEuler Intelligence 中的 RAG 技术能适应多种文档格式和内容场景，在不为系统增加较大负担的情况下，增强问答服务体验。



团队管理

团队管理是 openEuler Intelligence 中的 RAG 技术的基础能力之一，其通 RBAC 的机制来管控团队内成员对角色、成员和资产的访问，以增强知识库整体的易用性：

对于角色：涉及对角色原子权限构成的编辑、角色的增删改和成员角色的赋予，这些能力的原子化是实现团队权限实例化的核心。

对于成员：涉及对成员的邀请和申请请求的处理以及成员的增删改查，这些能力的原子化是实现管理成员进组及其权限赋予的核心。

对于资产：涉及对资产库和资产库内的文档、数据集和测试结果的增删改查，这些能力的原子化是实现资产数据按需访问的核心。

语料治理

语料治理是 openEuler Intelligence 中的 RAG 技术的基础能力之一，其通过上下文位置信息提取、文本摘要和 OCR 增强等方式将语料以合适形态入库，以增强用户查询命中期望文档的概率：

- 上下文位置信息提取：对于文档内容，留存文档相对位置关系，包过片段的全局相对偏移和局部相对偏移，为上下文补全提供基础数据；
- 文本摘要：对复杂文档或者片段，通过滑动窗口+大模型对文本进行摘要，为后续多层次检索的方式提供基础数据；

- OCR 增强：对图文混合的文档，基于图片文字内容+图片上下文进行摘要，为后续针对图片的提问提供基础数据；
- 文档溯源：openEuler Intelligence 在生成答案的过程中，基于返回的片段和相关的文档信息，在对应的句子右下方生成角注，点击角注可以获取文档的名称和摘要，增加问答生成的可信度。

自动化测试

自动化测试是 openEuler Intelligence 中的 RAG 技术的基础能力之一，其通过自动化数据集生成和测试评估识别知识库和检索增强算法等配置的不足：

- 数据集生成：用户选择解析完成的文档，基于用户选择的文档，随机选择部分解析结果，并将这些解析结果发送给大模型，让大模型生成及过滤问答对，最终形成含有查询、标准答案和原始片段的高质量测试数据集；
- 测试评估：基于用户生成的数据集和配置的检索增强算法、大模型、topk 片段限制等参数展开测试，最终基于测试集中的查询、标准答案、原始片段、关联到的片段和大模型拟合结果进行打分，最终使用精确率、召回率、忠实值、可解释性、最长公共子串得分、编辑距离得分和杰卡德相似系数对测试结果进行评估。

应用场景

- 面向 openEuler 普通用户：基于 openEuler 社区文献例如白皮书或者用户本地文档，可以构建定制化智能助手。
- 面向 openEuler 开发者：基于 web 端，开发者可以构建自己的工作流或者 Agent 应用，降低一些场景（代码审计等）重复劳动。
- 面向 openEuler 运维人员：基于 shell 端，运维人员可以通过自然语言与 os 交互，降低复杂命令构造和修改成本。

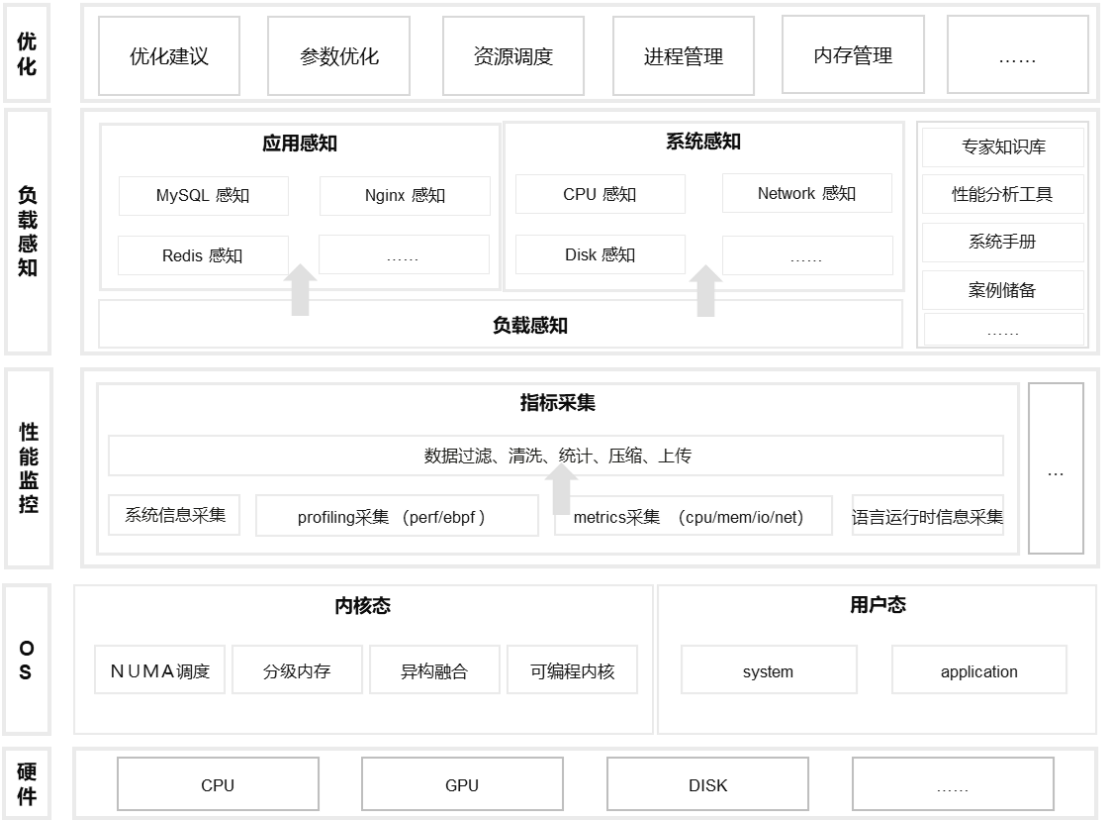
相关使用方式请参考 openEuler-intelligence-专区

智能调优

功能描述

openEuler Intelligence 智能调优功能目前支持智能 shell 入口。

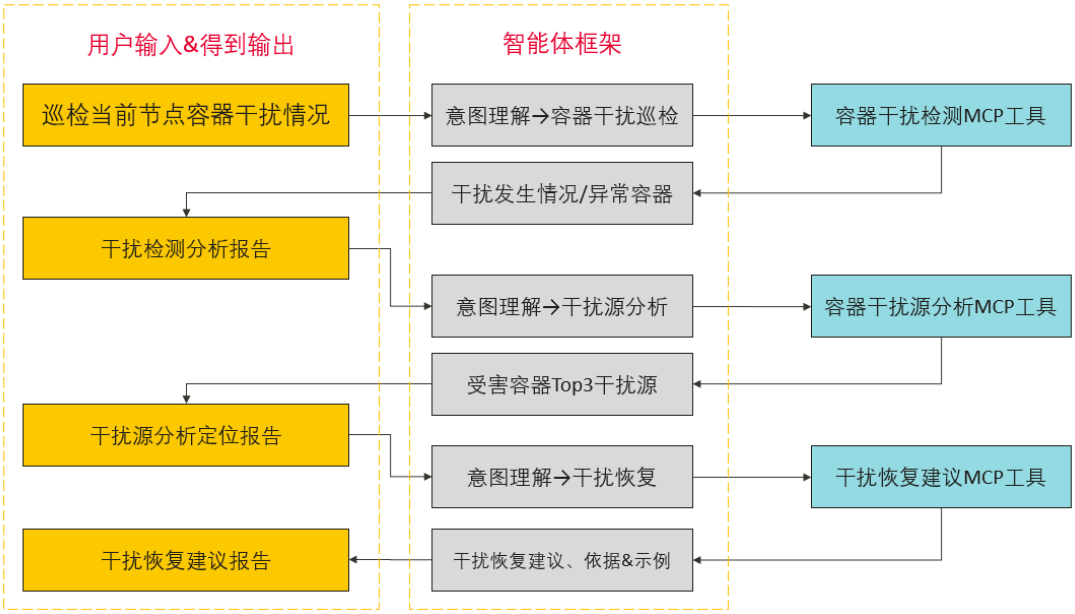
在上述功能入口，用户可通过与 openEuler Intelligence 进行自然语言交互，完成性能数据采集、系统性能分析、系统性能优化等作业，实现启发式调优，已支持通过 MCP 协议实现调优意图识别。



应用场景

- 快速获取系统重要性能指标数据：可快速获取当前系统中 CPU/IO/DISK/NETWORK/微架构等多个重要维度的性能指标以及指定应用的性能指标，帮助用户快速了解系统性能数据。
- 分析系统性能状况：可生成性能分析报告，报告从 CPU/IO/DISK/NETWORK/微架构等多个重要维度分析系统性能状况以及分析指定应用的性能状况，并提示当前系统可能存在的性能瓶颈。
- 推荐系统性能优化建议：可针对当前系统性能情况推荐一组合适的应用或系统参数，并自动使能推荐参数配置，可生成一键式执行的性能优化脚本，用户在审核脚本内容后，可执行该脚本，对系统及指定应用的配置进行优化。

智能诊断



功能描述

1. 干扰检测巡检：调用容器干扰检测工具，对节点中存在监控的节点在指定的时间段内的干扰进行检测，报告节点中存在的干扰情况，报告存在干扰的容器和对应指标。
2. 干扰源分析：调用容器干扰源分析工具，对节点中存在的所有受干扰容器进行分析，报告各个受干扰容器的 Top3 干扰源容器和对应的关联指标（如 CPU run delay）。
3. 干扰恢复建议生成：根据干扰源分析定位结果，调用干扰恢复建议工具，对存在的干扰进行分析并给出干扰恢复建议报告，提供各条建议的依据和示例指令。

应用场景

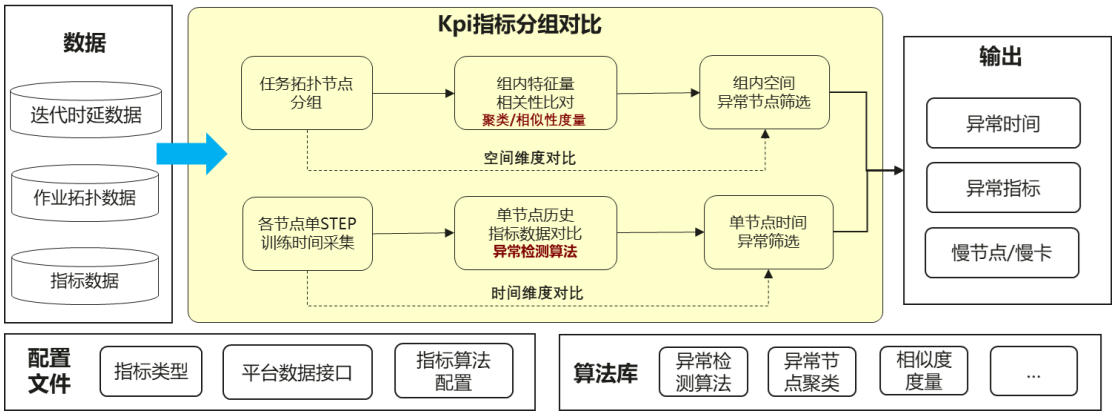
智能诊断接口在本次 openEuler 24.03 LTS SP3 版本的功能范围是：容器干扰检测、干扰源分析、干扰恢复建议生成。

- 其中容器干扰检测指的是：对单机上的已有监控的容器指标进行干扰检测，报告受干扰容器。
- 其中干扰源分析指的是：结合干扰检测结果和节点中的监控指标，进行干扰源分析，输出各个受干扰容器的 top3 根因指标。
- 其中干扰恢复建议生成指的是：基于干扰源分析结果，生成对应干扰恢复建议报告。

AI 集群慢节点定界

AI 集群在训练过程中不可避免会发生性能劣化，导致性能劣化的原因很多且复杂。现有方案是在发生性能劣化之后利用日志分析，但是从日志收集到问题定界根因诊断以及现网闭环问题需要长达 3-4 天之久。基于上述痛点问题，我们设计了一套在线慢节点定界方案，该方案能够实时在线观测系统关键指标，并基于模型和数据驱动的算法对观测数据进行实时分析给出劣慢节点的位置，便于系统自愈或者运维人员修复问题。

功能描述



基于分组的指标对比技术提供了 AI 集群训练场景下的的慢节点/慢卡检测能力。这项技术通过 sysTrace 实现，新增内容包括配置文件、算法库、慢节点空间维度对比算法和慢节点时间维度对比，最终输出慢节点异常时间、异常指标以及对应的慢节点/慢卡 ip，从而提高系统的稳定性和可靠性。该特性主要功能如下：

- 配置文件：主要包括待观测指标类型、指标算法配置参数以及数据接口，用于初始化慢节点检测算法。
- 算法库：包括常用的时序异常检测算法 spot 算法，k-sigma 算法，异常节点聚类算法和相似度度量算法。
- 数据：采集到的各个节点的指标数据，以时序序列表示。
- 指标分组对比：包括组内空间异常节点筛选和单节点时间异常筛选。组内空间异常节点筛选根据异常聚类算法输出异常节点；单节点时间异常筛选根据单节点历史数据进行时序异常检测判断节点是否异常。

应用场景

sysTrace 支持慢节点异常检测，告警展示，异常信息落盘。

AI 模型训练场景：适用于 AI 模型大规模集群训练任务，通过该功能可快速发现慢节点，便于系统自愈或者运维人员修复问题。

AI 模型推理场景：适用于单一模型的多实例性能劣化检测，通过多实例间应用资源的对比情况，可快速发现性能劣化的实例，便于推理作业的调度和资源利用率的提升。

智能容器镜像

功能描述

openEuler Intelligence 目前支持通过自然语言调用环境资源，在本地协助用户基于实际物理资源拉取容器镜像，并且建立适合算力设备调试的开发环境。

当前版本支持三类容器，并且镜像源已同步在 dockerhub 发布，用户可手动拉取运行：

1. SDK 层：仅封装使能 AI 硬件资源的组件库，例如：cuda、cann 等。
2. SDK + 训练/推理框架：在 SDK 层的基础上加装 tensorflow、pytorch 等框架，例如：tensorflow2.15.0-cuda12.2.0、pytorch2.1.0.a1-cann7.0.RC1 等。
3. SDK + 训练/推理框架 + 大模型：在第 2 类容器上选配几个模型进行封装，例如 llama2-7b、chatglm2-13b 等语言模型。

当前支持的容器镜像汇总：

registry	repository	image_name	tag
docker.io	openeuler	cann	8.0.RC1-oe2203sp4
			cann7.0.RC1.alpha002-oe2203sp2
docker.io	openeuler	oneapi-runtime	2024.2.0-oe2403lts
docker.io	openeuler	oneapi-basekit	2024.2.0-oe2403lts
docker.io	openeuler	llm-server	1.0.0-oe2203sp3
docker.io	openeuler	mlflow	2.11.1-oe2203sp3
			2.13.1-oe2203sp3
docker.io	openeuler	llm	chatglm2_6b-pytorch2.1.0.a1-cann7.0.RC1.alpha002-oe2203sp2
			llama2-7b-q8_0-oe2203sp2
			chatglm2-6b-q8_0-oe2203sp2
			fastchat-pytorch2.1.0.a1-cann7.0.RC1.alpha002-oe2203sp2
docker.io	openeuler	tensorflow	tensorflow2.15.0-oe2203sp2

			tensorflow2.15.0-cuda12.2.0-devel-cudnn8.9.5.30-oe2203sp2
docker.io	openeuler	pytorch	pytorch2.1.0-oe2203sp2
			pytorch2.1.0-cuda12.2.0-devel-cudnn8.9.5.30-oe2203sp2
			pytorch2.1.0.a1-cann7.0.RC1.alpha002-oe2203sp2
docker.io	openeuler	cuda	cuda12.2.0-devel-cudnn8.9.5.30-oe2203sp2

应用场景

- 面向 openEuler 普通用户: 简化深度学习开发环境构建流程, 节省物理资源的调用前提, 比如实现在 openEuler 系统上搭建昇腾计算的开发环境。
- 面向 openEuler 开发者：熟悉 openEulerAI 软件栈，减少组件配套试错成本。

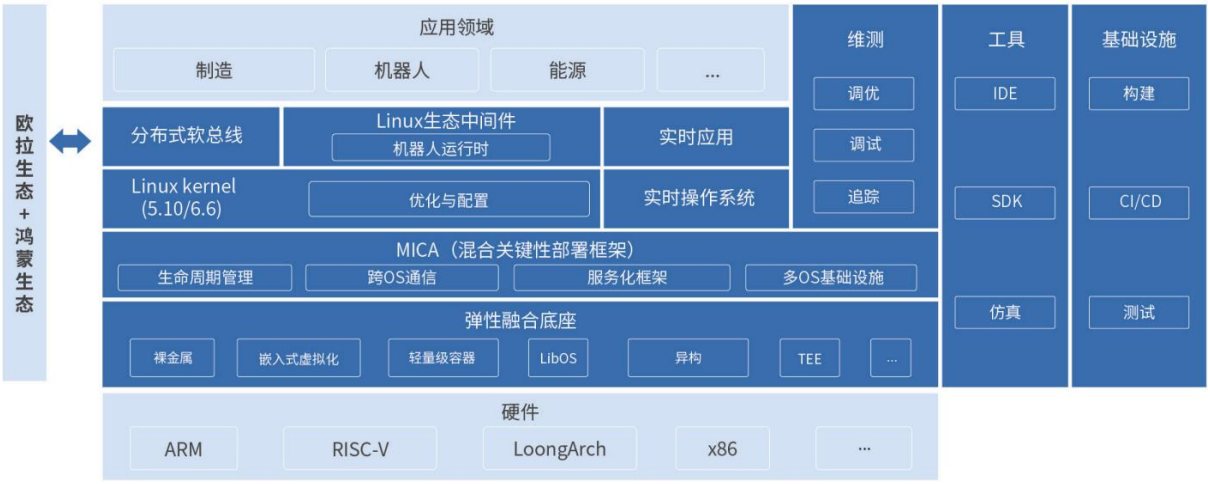
嵌入式场景创新

openEuler 发布面向嵌入式领域的版本 openEuler 24.03 LTS SP3，构建了一个相对完整的综合嵌入系统软件平台，在南北向生态、关键技术特性、基础设施、落地场景等方面都有显著的进步。

openEuler Embedded 围绕以制造、机器人为代表的 OT 领域持续深耕，通过行业项目垂直打通，不断完善和丰富嵌入式系统软件栈和生态。在软件包生态方面，回合了 oebridge 特性, 支持在线一键安装 openEuler 镜像仓软件包, 并支持在 Yocto 镜像构建时通过 oebridge 直接安装 openEuler RPM 包快速实现镜像定制。此外，还扩展支持了 oedeploy 特性，能够快速完成 AI 软件栈、云原生软件栈的部署。在内核支持方面，持续完善了 meta-openeuler 的内核配置，配合 oeaware 实时调优功能实现干扰控制以增强系统实时性。

未来 openEuler Embedded 将协同 openEuler 社区生态伙伴、用户、开发者，逐步扩展支持龙芯等新的芯片架构和更多的南向硬件，完善工业中间件、嵌入式 AI、嵌入式边缘、仿真系统等能力，打造综合嵌入式系统软件平台解决方案。

系统架构图



南向生态

openEuler Embedded Linux 当前主要支持 ARM64、x86-64、ARM32、RISC-V 等多种芯片架构，未来计划支持龙芯等架构，从 24.03 版本开始，南向支持大幅改善，已经支持树莓派、海思、瑞芯微、瑞萨、德州仪器、飞腾、赛昉、全志等厂商的芯片。

嵌入式弹性虚拟化底座

openEuler Embedded 的弹性虚拟化底座是为了在多核片上系统（SoC, System On Chip）上实现多个操作系统共同运行的一系列技术的集合，包含了裸金属、嵌入式虚拟化、轻量级容器、LibOS、可信执行环境（TEE）、异构部署等多种实现形态。不同的形态有各自的特点：

1. 裸金属：基于 openAMP 实现裸金属混合部署方案，支持外设分区管理，性能最好，但隔离性和灵活性较差。目前支持 UniProton/Zephyr/RT-Thread 和 openEuler Embedded Linux 混合部署。
2. 分区虚拟化：基于 Jailhouse 实现工业级硬件分区虚拟化方案，性能和隔离性较好，但灵活性较差。目前支持 UniProton/Zephyr/FreeRTOS 和 openEuler Embedded Linux 混合部署，也支持 openHarmony 和 openEuler Embedded Linux 的混合部署。
3. 实时虚拟化：openEuler 社区孵化了嵌入实时虚拟机监控器 ZVM 和基于 rust 语言的 Type-I 型嵌入式虚拟机监控器 Rust-Shyper，可以满足不同场景的需求。

混合关键性部署框架

openEuler Embedded 打造了构建在融合弹性底座之上混合关键性部署框架，并将其命名为 MICA（MIxed CriticAlity），该框架旨在通过一套统一的框架屏蔽下层弹性底座形态的

不同，从而实现 Linux 和其他 OS 运行时便捷地混合部署。依托硬件上的多核能力使得通用的 Linux 和专用的实时操作系统有效互补，从而达到全系统兼具两者的特点，并能够灵活开发、灵活部署。

MICA 的组成主要有四大部分：生命周期管理、跨 OS 通信、服务化框架和多 OS 基础设施。生命周期管理主要负责从 OS（Client OS）的加载、启动、暂停、结束等工作；跨 OS 通信为不同 OS 之间提供一套基于共享内存的高效通信机制；服务化框架是在跨 OS 通信基础之上便于不同 OS 提供各自擅长服务的框架，例如 Linux 提供通用的文件系统、网络服务，实时操作系统提供实时控制、实时计算等服务；多 OS 基础设施是从工程角度为把不同 OS 从工程上有机融合在一起的一系列机制，包括资源表达与分配，统一构建等功能。

混合关键性部署框架当前能力：

1. 支持裸金属模式下 openEuler Embedded Linux 和 RTOS（Zephyr/UniProton）的生命周期管理、跨 OS 通信。
2. 支持分区虚拟化模式下 openEuler Embedded Linux 和 RTOS（FreeRTOS/Zephyr）的生命周期管理、跨 OS 通信。

北向生态

1. 北向软件包支持：700+嵌入式领域常用软件包的构建。
2. 软实时内核：提供软实时能力，软实时中断响应时延微秒级。
3. 分布式软总线基础能力：集成 OpenHarmony 的分布式软总线和 hichain 点对点认证模块，实现 openEuler 嵌入式设备之间互联互通、openEuler 嵌入式设备和 OpenHarmony 设备之间互联互通。
4. 嵌入式容器与边缘：支持 iSula 容器，可以实现在嵌入式上部署 openEuler 或其他操作系统容器，简化应用移植和部署。支持生成嵌入式容器镜像，最小大小可到 5MB，可以部署在其他支持容器的操作系统之上。

UniProton 硬实时系统

UniProton 是一款实时操作系统，具备极致的低时延和灵活的混合关键性部署特性，可以适用于工业控制场景，既支持微控制器 MCU，也支持算力强的多核 CPU。目前关键能力如下：

- 支持 Cortex-M、ARM64、X86_64、riscv64 架构，支持 M4、RK3568、RK3588、X86_64、Hi3093、树莓派 4B、鲲鹏 920、昇腾 310、全志 D1s。

- 支持树莓派 4B、Hi3093、RK3588、X86_64 设备上通过裸金属模式和 openEuler Embedded Linux 混合部署。
- 支持通过 gdb 在 openEuler Embedded Linux 侧远程调试。

应用场景

openEuler Embedded 可广泛应用于工业控制、机器人控制、电力控制、航空航天、汽车及医疗等领域。

超节点场景创新

算力需求指数级增长与高速互联技术突破，驱动硬件从单节点向超节点加速演进，未来数据类、资源类、业务类的核心诉求从单机性能渐变到超节点，具有以下特点：

- 资源池化：计算、内存、互联和存储等均可池化，支持任意多对多协同。
- 规模扩展：按需灵活扩展，利用率高，大范围、整系统相同编址机制，控制静态传输、动态时延。
- 长稳可靠：10 倍提升长稳运行时间，自动避障，自动恢复，易部署，机房环境适应性强，易扩展，从模组到柜级均可扩展。

面对超节点调度、池化、通信、虚拟化等诉求，openEuler 异构融合系统升级，支持超节点形态，加速释放超节点异构算力。

超节点 OS 架构

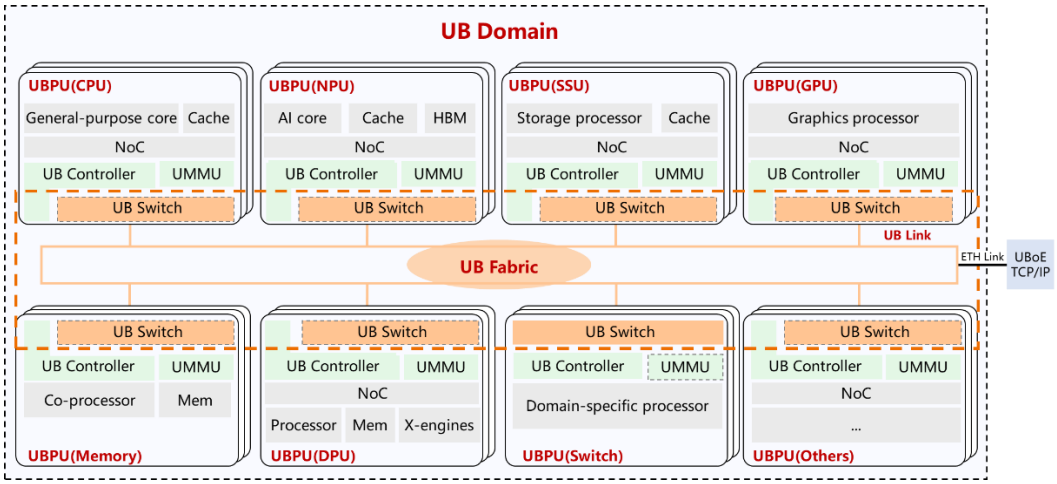
在超节点应用落地过程中，业界面临三大核心挑战：如何简化开发流程，实现业务“零修改”平滑迁移，如何精准匹配多样化算力供给，实现按需高效释放，如何应对系统复杂性提升，保障应用高可用性。openEuler 异构融合系统针对上述挑战方案如下：

- 第一，新增系统高阶服务能力，将业务迁移从“复杂适配”简化为“零修改适配”，大幅降低开发与迁移成本。
- 第二，强化异构融合核心子系统，提供异构融合通信与虚拟化能力，精准匹配算力需求，实现利用率最大化。
- 第三，完善池化设备管理体系，通过灵活扩展支持超节点形态，以池化高可靠管理防范故障扩散，筑牢业务稳定运行根基。

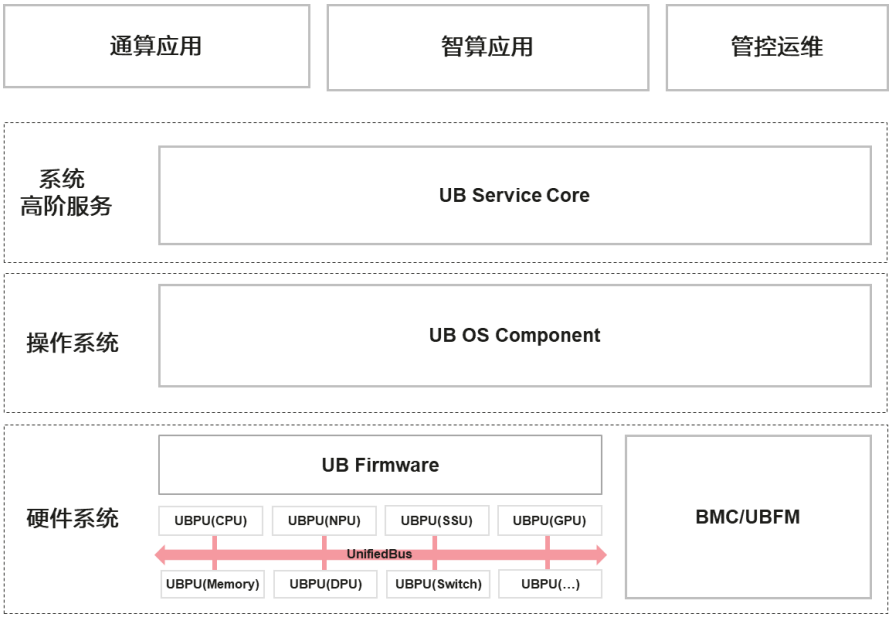
灵衢超节点 OS 实现方案

灵衢系统架构

超节点是灵衢计算系统的核心，通过灵衢重构计算系统，打破计算主机边界，实现智算和通算超节点扩展并获得性能规格收益，是面向未来 AI 时代的目标计算系统架构。灵衢超节点计算系统架构示意如下：



灵衢超节点打破传统主机边界，在超低时延、多协议归一的灵衢总线级互联的超节点域内，通过计算资源全量池化、平等互联，实现计算资源的灵活组合，超大规模组网和系统高可用性。灵衢计算系统整体参考架构如下图所示：



灵衢计算系统参考架构分成四层，共同发挥灵衢计算系统超节点架构优势：

1. **灵衢硬件系统，实现超节点可定义计算**

- UnifiedBus（UB/UB Firmware/BMC）：灵衢超节点计算系统基础，基于灵衢实现计算集群总线级互联、协议归一、平等协同、全量池化、大规模组网、高可用性。
- 灵衢互联结构管理器（UBFM）：完成灵衢的互联配置，实现计算资源的灵活、高效聚合，优化计算互联 SLA（时延、带宽、可靠性），实现灵衢超节点灵活算力使能。

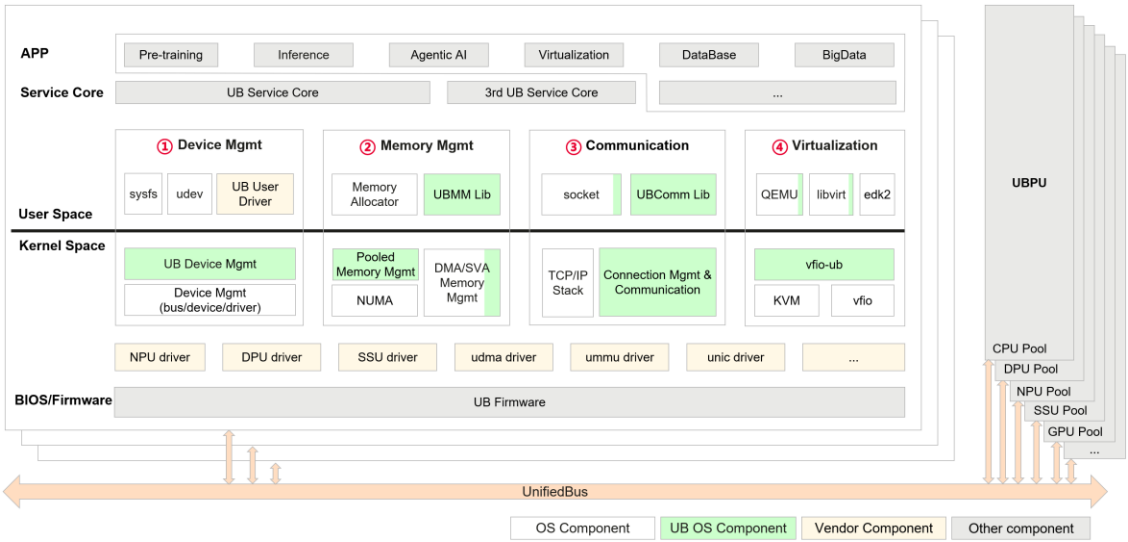
2. **操作系统灵衢组件（UB OS Component）完成灵衢和设备的使能，进行统一抽象与管理**

- UB OS Component 通过扩展 Linux 内核，原生支持超节点计算系统，实现对灵衢设备的统一抽象与管理。
- 同时保持 POSIX 接口兼容，实现现有应用的快速迁移。通过 openEuler 社区，使能操作系统生态，实现灵衢计算系统的开放创新。

3. **灵衢系统高阶服务（UB Service Core），使能超节点性能最优**

- 充分发挥灵衢计算系统在内存池化、通信、分布式操作等方面的独特优势，同时简化灵衢资源管理与调度的复杂性，提供了系列的灵衢系统高阶服务 UB Service Core，使能计算业务应用，快速发挥灵衢超节点系统优势，获得性能收益。

操作系统灵衢组件



操作系统灵衢组件（UB OS Component）是在 OS 原有内存管理、通信、设备管理和虚拟化框架上扩展支持灵衢，扩展的 4 个功能分别是：

1. **Device Mgmt：提供 UB 总线、UB 设备管理能力，实现计算节点内 UB 设备热插拔、配置。包含模块：**

- UB Device Mgmt：UB 设备管理基于 Linux 的 Device Mgmt（bus/device/driver）模型扩展支持 UB 设备管理，包含 UB 总线驱动、UB Firmware 交互、UB 设备热插拔等，同时提供设备驱动接口用于 UB 设备驱动开发。
- sysfs：UB 设备管理会在 sysfs 生成 UB 设备、驱动、总线信息，用户可通过 sysfs 查看和使用。
- udev：UB 设备管理会生成 uevent 事件，用户可通过已有 udev 工具获取 UB 设备的热插拔状态，实现对 UB 设备的管理。
- UB User Driver：UB 设备用户态驱动，用户可通过此接口将 UB 设备暴露到用户态，供用户态驱动或者虚拟化软件使用。

2. **Memory Mgmt：提供 UB 总线域内内存语义访问能力，实现跨计算节点跨设备内存借用、共享。包含模块：**

- DMA/SVA Memory Mgmt：基于 Linux 已有的 DMA/SVA 框架扩展实现对 UMMU 单元的管理，驱动或其他组件可通过本模块将主机内存注册给 UB 设备使用。
- Pooled Memory Mgmt：新增本地内存导出、远端内存导入、共享内存一致性维护等功能。基于 Linux 已有的 NUMA 内存管理框架上支持远端内存上线到 NUMA node 使用，或上线到内存设备（在/dev 下生成内存字符设备）通过 mmap 映射使用。
- UBMM Lib：用户态新增封装 Pooled Memory Mgmt 提供的接口，UBPRM 可通过上述接口实现 Home 和 User 侧的内存管理操作。
- Memory Allocator：兼容已有 glibc/jemalloc/tcmalloc 等内存管理接口，UBPRM 从其他计算节点借用内存并通过 UBMM Lib 上线到 NUMA node 后，应用程序无需修改即可通过 glibc/jemalloc/tcmalloc/llibnuma 等提供的 malloc/free/numa_alloc_onnode 等内存管理接口申请/释放借用的内存。

3. **Communication：提供跨计算节点、跨设备通信和远程调用功能。包含模块：**

- Connection Mgmt & Communication：实现异步通信的连接管理，同时提供内核态的异步通信接口供其他模块使用。
- UBComm Lib：新增提供灵衢异步通信和统一远程过程调用接口，应用程序可通过此接口实现超节点内节点间通信和函数调用。
- socket：兼容已有 glibc 等提供 socket 接口的库，应用程序无需修改或少量修改即可通过 socket 接口实现两个节点间通信。

4. **Virtualization：提供 UB 设备直通虚拟机能力。包含模块：**

- qemu：在现有 qemu 上扩展实现 Bus Controller、UB 设备、UMMU 的虚拟化。
- libvirt：在现有 libvirt 上扩展支持 UB 设备的创建、删除等功能。
- vfio-ub：基于 Linux 已有 vfio 框架上新增支持 UB 设备虚拟化。

操作系统灵衢组件的功能介绍和代码介绍见灵衢社区《灵衢®使能操作系统参考设计》。

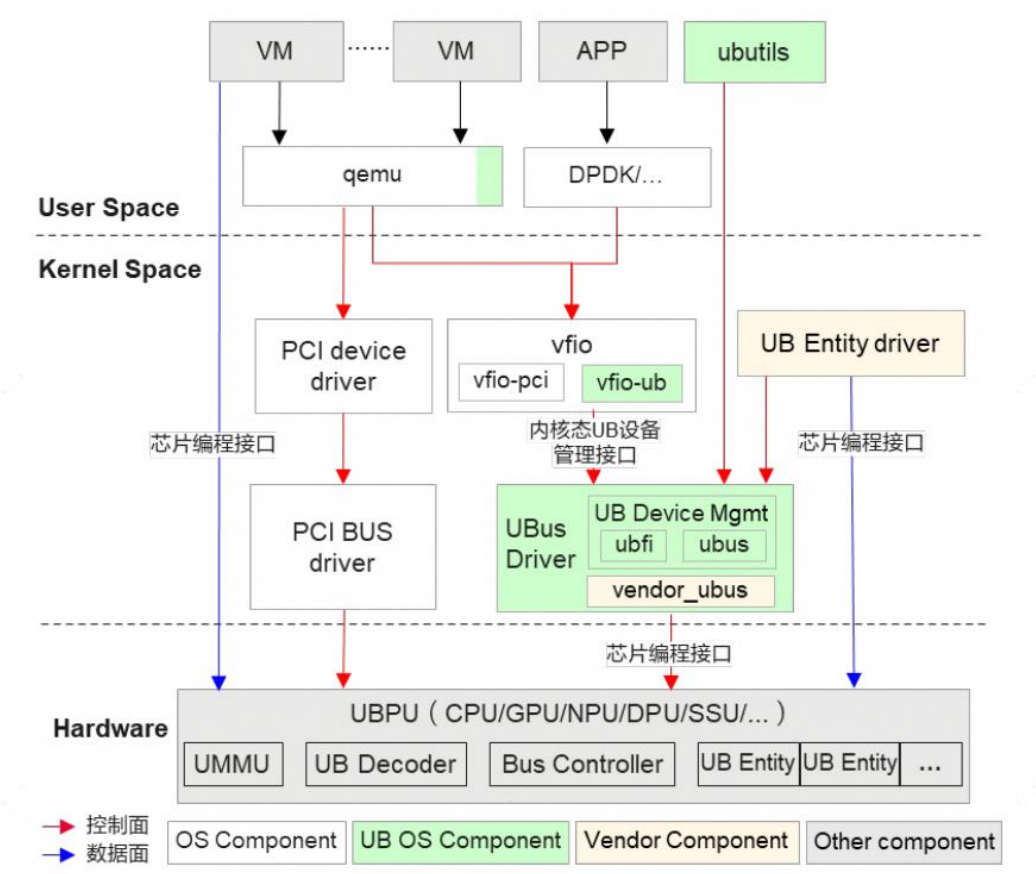
灵衢设备管理

UB 设备管理包括 UBus Driver、vfio-ub 和 ubutils 三个模块，对用户提供 UB 设备管理接口，包括给 UB 设备驱动提供了基本的设备发现、设备注册、中断使能等总线服务，UB

设备直通用户态的服务，以及给用户提供查询 UB 设备信息和配置服务。各类 UB 设备可以注册到 UB 总线上，对用户提供相应的功能。

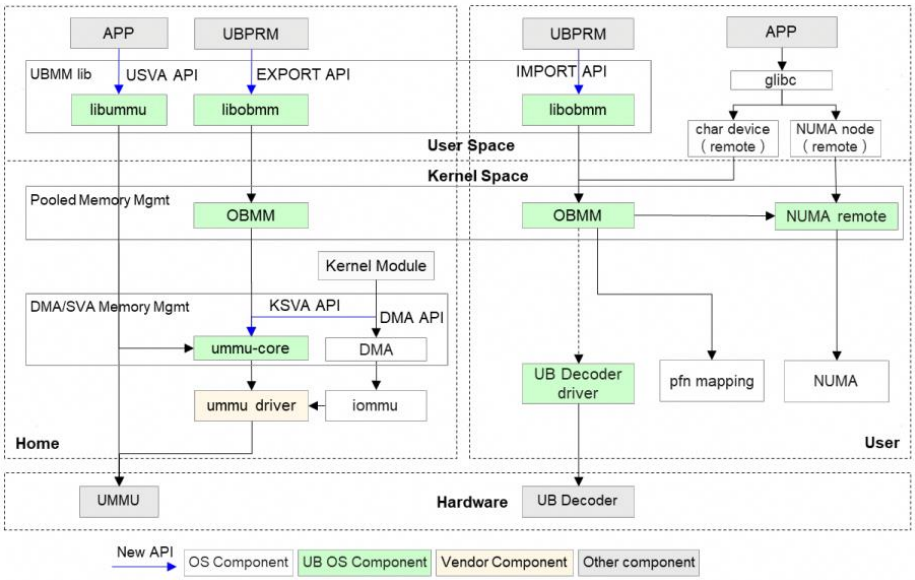
在 OS 内，UB 设备的发现、注册、驱动加载均由设备管理实现，分单机和集群场景：

1. 单机场景：单台服务器主机独立工作，每台服务器独享连接在主机上的所有 UB 设备，单机上的 UBus Driver 完成所有 UB 设备的枚举发现和使能，配合 vfio-ub 提供 UB 设备直通用户态能力。
2. 集群场景：超节点形态下 UBus Driver 提供分配给计算节点上的 UB Entity 发现和使能，并支持池化设备接入和移除，配合 vfio-ub 提供 UB 设备直通用户态能力。



灵衢内存管理

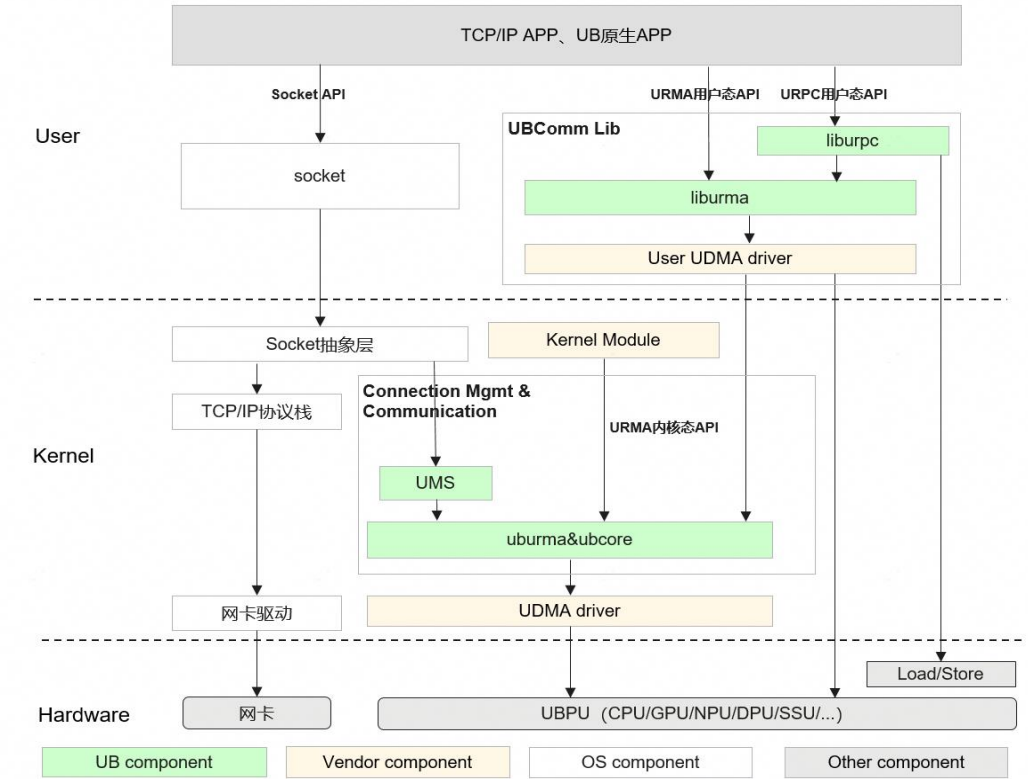
现有总线采用 Host-Device 模型，以 Host（CPU）为中心，Host 管理每一个 Device。在这个模型之下，Host 访问 Device 内存、Device 访问 Host 内存、Device 访问 Device 内存共三类访存路径有单独实现。UB 统一上述访存模型，根据一个访存发起流程将相关的节点分为 User 和 Home，User 可以通过同步（Load/Store）或者异步（DMA）的方式访问 Home 侧内存资源。



基于 UB 总线的能力，OBMM 模块提供了节点间数据通路配置与跨节点数据一致性维护能力。完成配置后，用户可以在 UB 互联的集群中，在一个节点上访问另一个节点上的内存，实现跨节点内存共享、内存池化与节点间借用等功能。

灵衢通信

灵衢通信库是分布式通信软件库，为数据中心网络、超节点内、服务器内的卡与卡之间提供高性能的通信接口，使能和释放 UB 硬件能力。



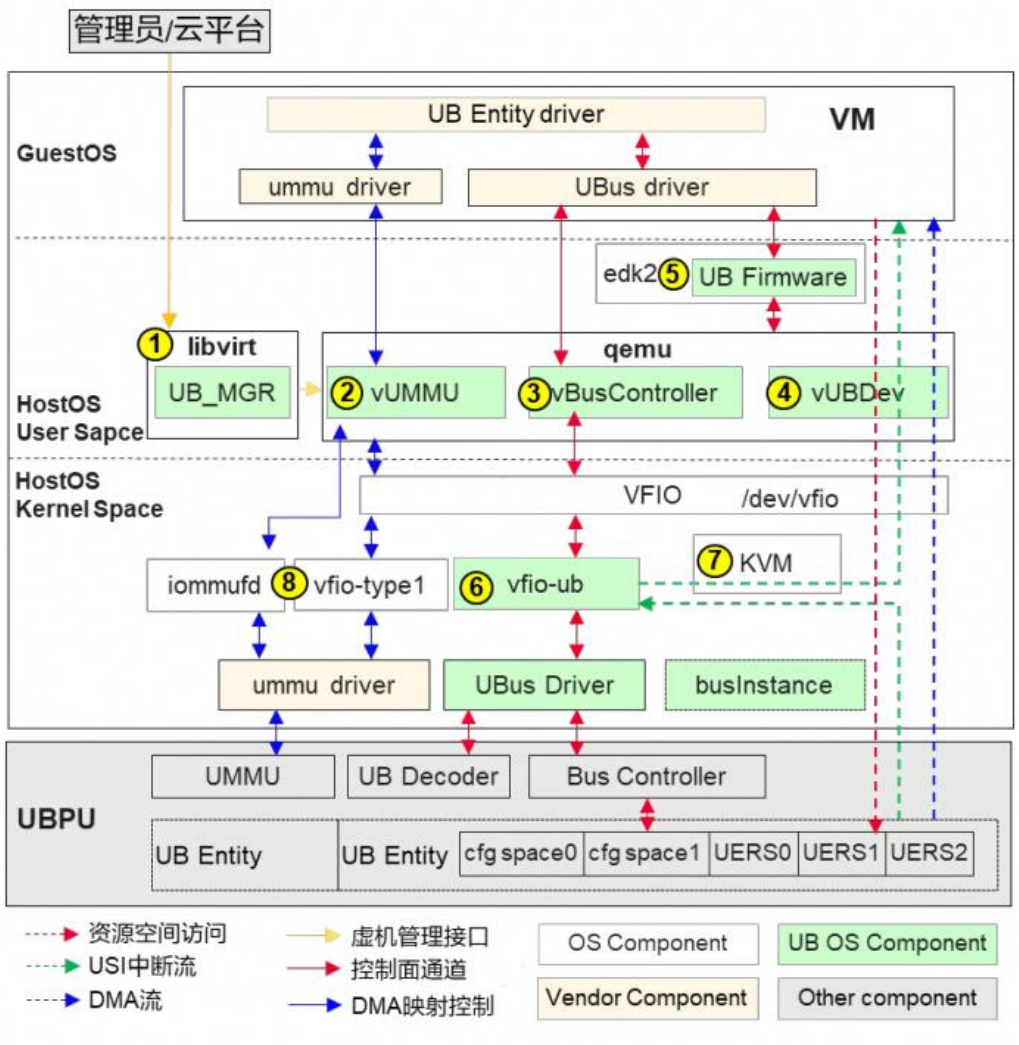
- UMS：UMS 支持对接 Socket 抽象层，是北向兼容 Socket 编程接口，南向基于灵衢高性能网络进行数据传输的内核网络协议栈，透明加速 TCP 应用，实现性能提升。
- URMA：URMA 是灵衢通信的基础软件库，屏蔽硬件差异，基于灵衢高性能网络提供读写、收发、原子操作等远端内存访问语义，是灵衢通信应用的基础。
- URPC：URPC 是灵衢原生的统一远程过程调用，支持灵衢原生高性能主机间和设备间 RPC 通信，以及 RPC 加速。

灵衢设备虚拟化

在云计算场景中，通过虚拟化将物理机上的硬件资源隔离给多个虚拟机，极大提升了资源利用率。UB 作为新一代高速互联总线，同样支持虚拟化能力。

当前设备虚拟化的主流技术主要包括如下几种：IO 全虚拟化、IO 半虚拟化、硬件辅助 IO 虚拟化。传统架构下每一台服务器都是通过插在其上的网卡连接到 TOR 交换机，所以最大可支持的带宽是固定的，在实际应用过程中会带来闲时带宽利用率低、忙时带宽不够用、服务器间带宽压力不均衡等问题。

在 UB 总线中，UE（UB Entity）作为一个 UB 设备相对隔离的功能单元，UB 虚拟化的功能同样支持通过直通的方式将其直接分配给虚拟机使用，以达到设备原生的性能，UB 设备虚拟化在传统的硬件辅助 IO 虚拟化之上扩展对池化 UB 设备的直通访问。通过将网卡、DPU 设备资源池化，可以解决上述问题。所有设备资源统一在设备池中，所有服务器通过共用设备池里的网卡资源，可以根据服务器实际的网络负载情况动态申请使用相对应的网卡资源，达到提升设备资源利用率，避免网络带宽瓶颈。同时，灵衢特有的内存管理与大带宽通信特性，能给虚机提供更灵活的内存超分与极速热迁移能力。

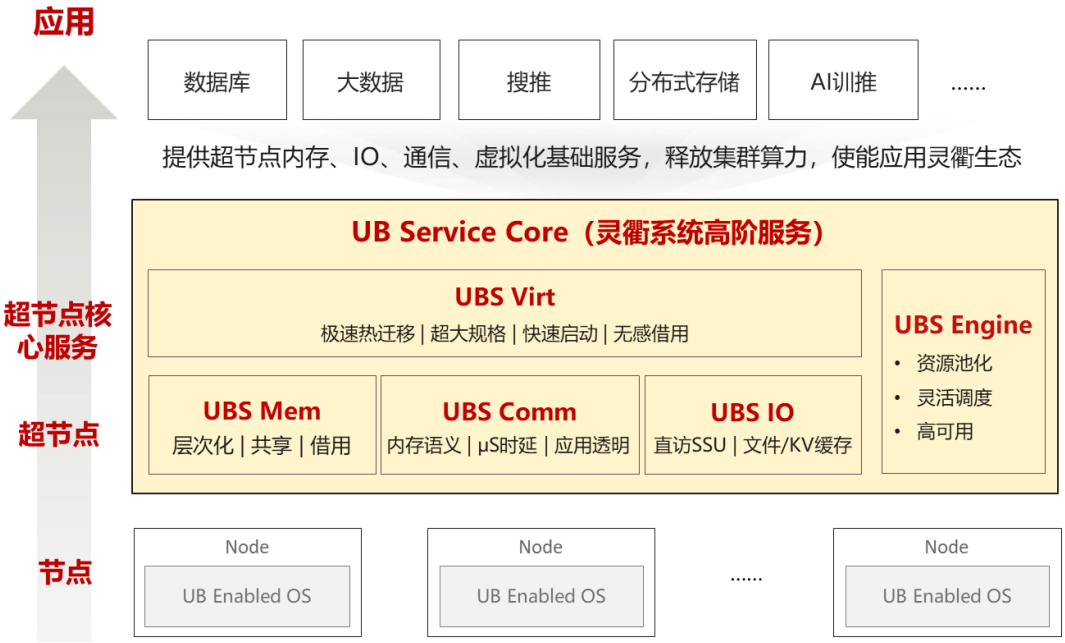


- UBNative: 基于 UB 协议构建的高性能设备虚拟化解决方案，通过模拟 UB 设备模型，实现虚拟机对 UB 总线的无缝接入。依托 VFIO 技术，将 UB 设备以直通方式给虚拟机使用，确保在虚拟化环境中充分发挥设备原生性能，实现近硬件级的运行效率与资源利用率，为高吞吐、低延迟场景提供稳定支撑。
- 内存超分：随着主机内存规格的增大，内存成本在整体 TCO 占比持续走高，云场景下提升内存利用率的目标越来越重要。实现基于灵衢超节点架构下的内存超分功能，构建面向大页虚拟机、直通虚拟机的内存回收方案，充分回收虚拟机内部的无用内存，转换让其发挥业务价值，从而能提升虚拟化内存利用率，进一步降低云厂商 TCO。
- 极速热迁移: 基于 URMA 内存读写语义，利用灵衢高性能网络进行内存数据迁移，减少迁移中断时间、减少资源开销、并提升高内存压力虚拟机的迁移成功率。

灵衢系统高阶服务

面向灵衢超节点，为了应用快速使能超节点能力，灵衢系统构建了 UB Service Core，封装 UB 底层能力、集群拓扑等，简化并兼容现有生态，让应用像使用本地资源一样使用超节点资源，简化应用开发，使能灵衢超节点。

UB Service Core 构筑 5 大集群级系统服务，释放超节点平等互联架构优势，全面使能应用加速 30~50%，促进灵衢系统软件生态构筑，灵衢系统高阶服务参考架构如下图所示：



UB Service Core (简写 **UBS Core**），包含如下 5 大部分：

1. **UB Service Core Engine**（简写 **UBS Engine**）：支持内存、DPU 资源池化管理与动态调度，支持分布式自选主，支持 N 节点场景下最多 N-1 节点失效的高可用，是灵衢计算系统的控制面核心参考实现。
2. **UB Service Core Memory**（简写 **UBS Mem**）：支持统一内存编程，实现灵衢超节点的共享内存、池化内存。
3. **UB Service Core Communication**（简写 **UBS Comm**）：基于超节点提供高性能、高可靠以及生态兼容（用户态 Socket/Verbs over UB）的通信协议。
4. **UB Service Core IO**（简写 **UBS IO**）：基于超节点，提供应用亲和的全局数据读写缓存系统高阶 IO 服务。
5. **UB Service Core Virt**（简写 **UBS Virt**）：支持虚拟化池化，热迁移策略决策，极速快恢与容灾，虚机/容器间极速通信等能力，使能虚拟化性能提升。

详情请见灵衢社区《灵衢®系统高阶服务软件架构参考设计》。

注：本次开源包含 UBS Engine、UBS Comm，其他预计于 2026 年开源。

应用场景

以灵衢为基础构建的超节点，在面向人工智能计算与通用计算领域的 10 大核心业务场景，如大模型预训练、中心推理、后训练与强化学习、多模态内容理解与生成、Agentic AI、虚拟化、大数据、数据库、分布式存储和高性能计算等，均可提供领先的系统能力，带来计算业务性能和资源利用率提升。

详情请见灵衢社区《基于灵衢®的超节点参考架构白皮书》。

5. 内核创新

openEuler 内核中的新特性

openEuler 24.03 LTS SP3 基于 Linux Kernel 6.6 内核构建，在此基础上，同时吸收了社区高版本的有益特性及社区创新特性。

- 文件系统支持可编程页缓存：针对大模型推理场景中模型加载 I/O 效率低下问题，实现一种页缓存可编程框架。该框架以透明化方式堆叠于当前文件系统之上，将文件系统缺页事件转发到用户态，使应用可以根据负载特性在用户态对文件系统的缓存策略进行定制，从而对不同模型的加载进行 I/O 效率的显著优化。
- 跨进程零拷贝数据传输机制：提供高效的节点内进程间数据传输接口，应用可将源进程中的给定虚拟内存地址空间上关联的页面映射到目的进程中的虚拟地址空间上，兼容 PMD 大页和 PTE 小页映射。映射后的目的地址访问权限和源地址保持一致。
- FUSE 文件系统支持 io_uring 通信接口：当前 FUSE 架构中，用户态守护进程与内核态驱动模块通过字符设备/dev/fuse 进行通信存在性能瓶颈。主要体现在三个方面：1. 多线程环境中，都需要通过/dev/fuse 设备获取 fuse 请求，容易出现锁争用情况；2. 当前 I/O 请求采用逐条传递机制，缺乏批量处理能力，导致 I/O 密集型任务效率较低；3. 发起读写的用户程序和实现文件读写的后端 fuse 线程通常在不同 CPU 核上，会带来较多的核心切换，降低效率。因此，采用 Fuse over io_uring 技术突破上述瓶颈，具体的方案有：1. 通过使用 io_uring 通信接口替代/dev/fuse，提高并

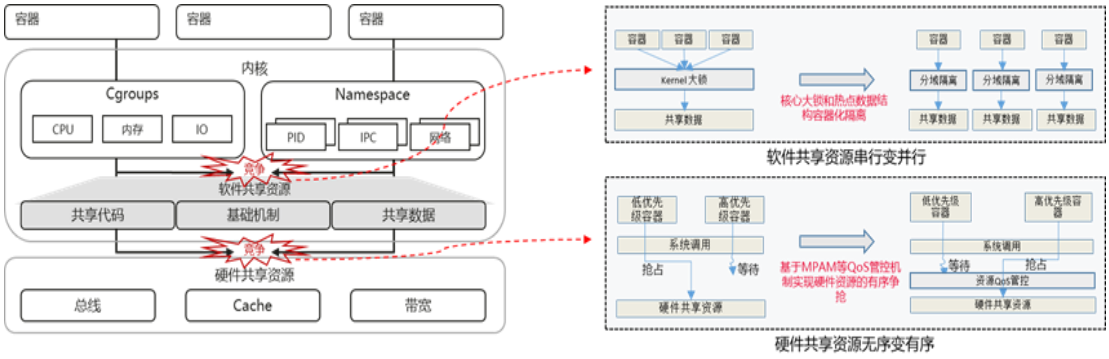
发能力；2. 后端 fuse 服务可以灵活选择走 io_uring 或者传统的/dev/fuse 字符设备通信。

- IO 混部场景 IO QoS 控制器 IOInflight：详情见众核高密。
- 基于 Xcall 的 epoll_wait 异步预取：面向特定系统调用的性能优化场景，Dynamic Xcall 通过劫持机制执行定制系统调用，允许用户在不侵入式修改内核的情况下，以应用程序 ELF 文件的粒度进行系统调用劫持，执行定制化系统调用。基于该框架实现了一套非内核侵入式修改的 epoll 异步预取示例内核模块，可以在 Redis 等场景获得一定的性能收益。

6. 云化基座

众核高密

服务器芯片由多核进入众核时代（>256C），对操作系统提出新的挑战。提升 Rack 计算密度、降低数据中心 TCO，众核服务器已成为互联网行业主流选择，随着云技术和业务规模发展，容器化部署成为互联网行业的主流业务部署形态，在这种场景下，系统串行开销和同步开销限制可扩展性，干扰问题凸显，资源利用率低，影响容器部署扩展性的串行访问开销和同步开销主要来自软硬共享资源争用。



功能描述

本期主要采用轻量虚拟化按 NUMA 分域拆分资源、域内实现资源容器级隔离增强，降低因软硬件资源争用导致的性能干扰，提升容器部署扩展性。关键技术特性如下：

- 虚拟机内存 QoS 控制：当多个租户的虚拟机（VM）部署于同一物理主机时，若内存密集型虚拟机占用大量内存带宽，可能引发资源争用，导致其他虚拟机无法获得足够的内存带宽以满足其业务性能需求，进而影响整体系统服务质量。基于鲲鹏处理器提供的内存带

宽监控与调控能力（MPAM, Memory Power and Access Monitoring），结合操作系统层面的 resctrl（Resource Control）机制，系统可实现对最多 30 个虚拟机的内存带宽使用情况进行精细化监测与动态控制。该能力支持对虚拟机内存带宽的上限、下限及优先级策略进行配置，从而构建多租户环境下的内存带宽资源隔离与保障体系。具体而言：内存带宽上限控制：通过为各虚拟机配置最大内存带宽使用阈值，有效防止单个虚拟机过度占用内存带宽资源，避免对其他租户虚拟机造成性能干扰；内存带宽下限保障：支持设定最低带宽保障值，确保在虚拟机实际负载较低时，系统可自动提升其带宽使用优先级，实现资源的动态优化与高效利用；优先级调度策略：支持基于业务重要性配置虚拟机的内存带宽优先级，优先保障关键业务虚拟机的带宽稳定供给，提升高优先级工作负载的可用性与服务质量；

- 虚拟设备 NUMA 亲和：PCI 设备也具有 Numa 亲和性，在主机侧直接访问，OS 调度系统会根据设备亲和性进行调度优化，以防止跨 Numa 访问 PCI 设备而造成性能损耗。对于虚拟机设备直通来说，当前还不具备在虚机内部呈现 PCI 设备亲和的 Numa 节点。本功能基于 PCI 扩展桥(PXB)扩展虚拟机 PCI 设备拓扑结构，支持在虚拟机内呈现虚拟设备所在 NUMA，便于系统 OS 优化调度或者用户根据虚拟设备所在 NUMA 部署业务应用，减少跨 NUMA 资源访问导致到性能损耗，提高虚拟机内业务应用性能；

- 轻量虚拟化：虚拟设备中断卸载实现虚拟 virtio 设备中断卸载至硬件注入，降低存储虚拟化损耗；PMD 队列负载均衡实现虚拟磁盘 IO 队列从单 reactor 均衡分散到多 reactor，消除 cpu 瓶颈，降低存储虚拟化损耗；

- CPU 分域调度：CPU 基于硬件拓扑划分子域部署容器，一个容器一个独立子调度域，实现容器之间干扰隔离，降低跨 cluster cache 同步次数和 cache/NUMA 内存等硬件资源争抢，减轻容器之间的相互干扰。对于 redis 多并发场景性能提升 10%+；

- 文件系统块分配干扰隔离：优化 ext4 块分配释放流程中的 group lock 和 s_md_lock 两个主要争抢的锁，以提高 EXT4 块分配流程的可扩展性。通过允许在当前目标块组被占用时尝试使用其他空闲块组进行分配，从而减少了多个容器争抢同一个块组造成的 CPU 浪费，并充分利用了 ext4 多块组的优势，缓解了 group lock 的竞争。其次通过将流式配的全局目标拆分为多个，从而减少了全局锁 s_md_lock 的竞争文件数据也更加聚集。在 64 容器并发场景下，块分配和块释放混合场景 OPS 提升 5 倍以上，单块分配场景提升 10 倍以上；

- 高效 slab 回收：将 slab 内存回收的读写锁，优化为 RCU 无锁化 slab 回收，不同 slab 之间的内存回收互不干扰，回收效率显著增加，多容器并发场景下，系统调用性能显著增强；

- 网络 tcp hash 干扰隔离：tcp_hashinfo bash、ehash 存在锁竞争，ehash 计算频繁，导致高并发下带宽下降，时延变大。将 tcp_hashinfo bash、ehash 的自旋锁改为 rcu，ehash 计算方式改为 lport 递增减少查询时间和计算次数，减少 tcp connect hash 的锁竞争；

- Cgroup 隔离增强：user namespace 通过 percpu counter 替换原来的原子操作，避免不同 namespace 相同父节点竞争访问，消除容器间 rlimit 计数干扰。 解决 will-it-scale/signal1 用例线性度问题，64 个容器并发吞吐性能提升 2 倍。通过对 memcg 实现批量释放处理，避免大量的小内存释放对于相同父节点计数竞争，提升内存计数的可扩展性，tlb-flush2 测试用例 64 容器吞吐提升 1.5 倍；基于 eBPF 可编程内核能力，提供主机容器信息隔离与过滤机制，高效实现容器资源视图。相较业界 LXCFS 方案，本方案避免了内核态-用户态切换开销，消除了 LXCFS 进程的性能与可靠性瓶颈，单容器内资源视图吞吐量在单容器场景下性能提升 1 倍，在 64 容器场景下提升 10 倍；

- 干扰监测：干扰监测回答的是容器有没有被干扰、是什么干扰、干扰程度如何这三个问题，从结果上看干扰可以分为干扰导致指令得不到执行、干扰导致指令执行变慢和干扰导致指令执行变多三类，干扰监测从内核角度，针对每一类的典型干扰在运行时进行统计，当前支持在线统计 schedule latency、throttling、softirq、hardirq、spinlock、mutex 和 smt 干扰，性能开销在 5%以内；

- 鲲鹏内存/Cache QoS 管控机制 MPAM：内存带宽流量和各级缓存占用量，可按照使用量上限/保底/优先级方式进行配置，根据不同业务，以线程为粒度部署不同隔离策略。支持业务资源实时监控，在客户业务层面和线程级别，实时对共享资源的使用情况进行跟踪监控，将资源使用情况反馈给控制策略，形成闭环控制效果。此外，MPAM 联动 SMMU 扩展外设 IO QoS 方案，支持对外围设备和异构加速器 IO 带宽流量进行隔离配置，按设备粒度级别进行资源监控；

- QoS 策略动态配置：提供集群级别的 mpam Qos 管理插件，基于 mpam 提供的 Qos 接口，在插件中根据用户定义，实现为所有节点自动分解各级别优先级，并根据用户的声明自动设置在离线任务 mpam Qos 优先级，从而实现在混部场景下对资源的充分利用：在线业务繁忙时自动抢占离线任务的 llc 及内存带宽，在线业务空闲时自动释放 llc 及内存带宽资源提升离线业务处理性能；

- IO 混部场景 IO QoS 控制器 IOInflight：在高密混部场景中，IO 资源争抢常导致在线业务干扰率高达 40% 以上，而传统的 blk-throttle 静态限流虽能降低干扰却严重浪费磁盘带宽。为此引入 IOInflight 控制器，通过基于时延监控与队列深度的动态节流机制，实现在线业务的精准抢占。该控制器将干扰率与底噪控制在 5% 以内，有效保障在线业务时延，同时使离线业务带宽较传统硬限制模式提升 2 倍。

应用场景

众核服务器场景下，业务容器高密度部署场景，通过降低容器间干扰，提升容器部署密度，进而提升资源利用率。

内存密集型场景：适用于多台虚拟机同时竞争带宽的场景，通过该功能可对每台虚拟机内存带宽占用进行监测和控制，保障对性能要求高的虚拟机业务能正常进行。

7. 开发者生态

DevStation 开发者工作站

DevStation 是基于 openEuler 的智能开发者工作站，专为极客与创新者而生。旨在提供开箱即用、高效安全的开发环境，打通从部署、编码、编译、构建到发布的全流程。它融合了一键式运行环境与全栈开发工具链，支持从系统启动到代码落地的无缝衔接。无需复杂安装，即可体验开箱即用的开发环境，通过新增 MCP AI 智能引擎，快速完成社区工具链调用，实现从基础设施搭建到应用开发的效率飞跃。

功能描述

开发者友好的集成环境：发行版预装了广泛的开发工具和 IDE，如 VS Codium 系列等。支持多种编程语言，满足从前端、后端到全栈开发的需求。

智能 CVE 修复与安全运维：深度集成 CVE 智能修复系统。该系统利用 LLM 分析、代码相似度算法（Levenshtein 距离）与自动化 back-porting 流程，智能识别并解决补丁应用中的代码冲突，在保留核心修复逻辑的前提下，将人工解决冲突的平均耗时从 4 小时降低至约 10 分钟，极大提升了安全漏洞的响应与修复效率。

社区原生工具生态：新增 oeDeploy（一键式部署工具）、epkg（扩展软件包管理器）、devkit 和 DevStation 智能助手，实现从环境配置到代码落地的全链路支持。 oeDevPlugin 插件+oeGitExt 命令行工具支持：专为 openEuler 社区开发者设计的 VSCodium 插件，提

供 Issue/PR 可视化管理面板，支持快速拉取社区代码仓、提交 PR，并实时同步社区任务状态。 DevStation 智能助手：支持自然语言生成代码片段、一键生成 API 文档及 Linux 命令解释。

图形化编程环境：集成了图形化编程工具，降低了新手的编程门槛，同时也为高级开发者提供了可视化编程的强大功能，预装 Thunderbird 等办公效率工具。

MCP 智能应用生态构建：Devstation 深度集成 Model Context Protocol (MCP) 框架，构建完整的智能工具链生态，预装 MCP 智能工具链，支持 oeGitExt、rpm-builder 等核心 MCP Server，提供社区事务管理、RPM 打包等能力，将传统开发工具（如 Git、RPM 构建器）通过 MCP 协议进行智能化封装，提供自然语言交互接口。并提供完整的 MCP 自动化转换

与质量保障工具链，能够基于社区现有工具和测试套（mugen）自动生成高质量、可测试的 MCP Server，并配套 mcp-testkit 测试框架 确保服务可靠性。

系统部署与兼容性增强：广泛的硬件支持，特别优化对主流笔记本/PC 硬件的兼容性（触摸板、Wi-Fi 、蓝牙），重构内核构建脚本（kernel-extra-modules），确保裸机部署体验。灵活部署形态，支持 LiveCD（一键运行无需安装）、裸机安装、虚拟机部署。

全新安装工具 heolleo：heolleo 是一款专为 DevStation 设计的现代化客户端工具。其核心使命是简化 DevStation 的安装流程。采用模块化设计使其可以轻松扩展以支持不同的硬件架构（如 x86/ARM）、文件系统或引导加载器（GRUB 等）。支持从本地 ISO 镜像、网络地址（HTTP/FTP）等快速获取系统文件，提供灵活的安装方式：

1. 本地 ISO 安装:对于追求极致稳定、速度或需要在无网络、受限环境中部署系统的用户，heolleo 提供本地 ISO 安装模式。充分利用已有的系统镜像文件，提供一个高速、可靠且完全离线的安装体验，当前已实现自动化分区安装。
2. 网络安装：heolleo 的网络安装模式适应现代系统部署的趋势。可直接从互联网上的服务器获取最新系统文件，省去了手动下载镜像的步骤，能以最便捷的方式触及最新的 DevStation 版本。

应用场景

操作系统安全与 CVE 高效修复: 面向安全团队与内核维护者, 提供智能 CVE 修复能力, 自动化处理补丁冲突, 将单补丁处理时间压缩至 10 分钟以内, 实现漏洞的快速响应与修复, 保障系统安全。

多语言开发环境：适用于需要同时开发多语言（如 Python、JavaScript、Java、C++）项目的开发者，无需手动配置环境，系统预装各种编译器、解释器和构建工具。

快速安装部署平台：Devstation 集成了 oeDeploy，可以对 kubeflow、k8s 等分布式软件实现分钟级部署，大幅减少开发者部署时间。oeDeploy 同时提供了统一的插件框架与原子化的部署能力，开发者只需遵循简单的开发规范，就可以快速发布自定义的安装部署插件，帮助更多用户解决安装部署问题。

面向测试/南向兼容性开发人员，提供硬件兼容性保障（支持主流笔记本/服务器），支持裸机部署测试驱动兼容性。

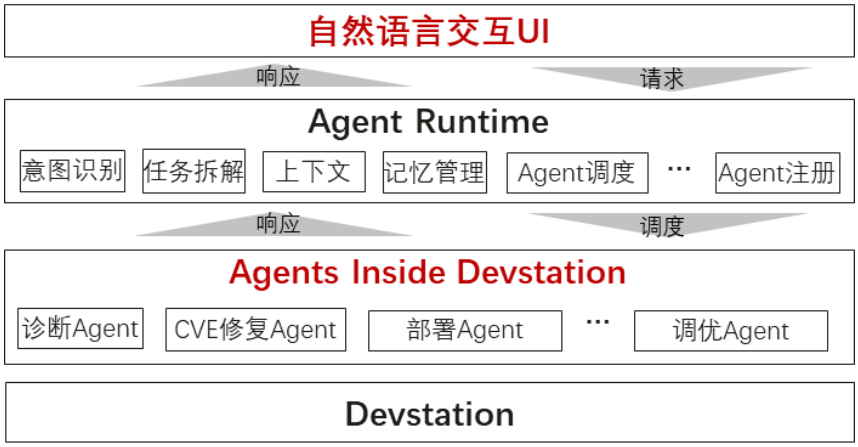
提升开发者效率：MCP RPM-builder 工具链落地，提升 MCP 易用性，支持 mcp servers 自动化打包构建 rpm 包并发布到社区，确保 mcp 一键安装功能 100%可用，构建完整的 MCP

智能应用生态 REPO，覆盖部署、测试、性能调优等应用场景，包括：查询分配的社区 issue，创建 PR 提交代码变更，通过 CI/CD 自动构建验证。

DevStation 智能助手（Polymind）

功能描述

DevStation 自带桌面 AI 助手软件，提供对话窗口，主动识别并理解用户的意图，支持用户通过自然语言的形势与操作系统交互，大幅降低开发者开发、运维和使用门槛。



交互 UI 层：DevStation 智能助手提供自然语言交互 UI，用户诉求会被传递到 Agent Runtime 进行处理。

Agent Runtime：作为整个智能助手的大脑，理解用户的意图并拆解转换成子任务，将任务委派给集成在 DevStation 中的 Agent、Mcp 执行，循环委派调度，直到所有子任务完成。

Agents Inside DevStation：依托于 openEuler 庞大的 Mcp、Agent 软件仓库，openEuler DevStation 正逐步演变成由 Agent 构成的 Agentic OS，当前 DevStation 智能助手除了内置的 Agent，还支持通过 DevStation 中的应用商店选择安装相关的 Mcp 和 Agent。

应用场景

DevStation 桌面 AI 助手以自然语言交互为核心，深度融入开发者全流程工作场景，凭借精准的意图识别能力，将复杂的系统操作与技术任务转化为简单的对话指令，全方位降低开发、运维及系统使用门槛，为开发者打造高效便捷的工作闭环。

开发者无需熟记复杂的环境配置命令，只需在对话窗口输入任务诉求，AI 助手便会主动识别需求，自动完成环境配置、服务部署、代码开发、测试、构建等需求。无论是新手开发

者快速上手、资深开发者提升效率，还是运维人员简化排查流程，DevStation 桌面 AI 助手都以自然、智能的交互方式，将操作系统的复杂能力转化为可直接对话的服务，让开发者无需关注底层技术细节，更专注于核心业务研发。

部署工具 oeDeploy

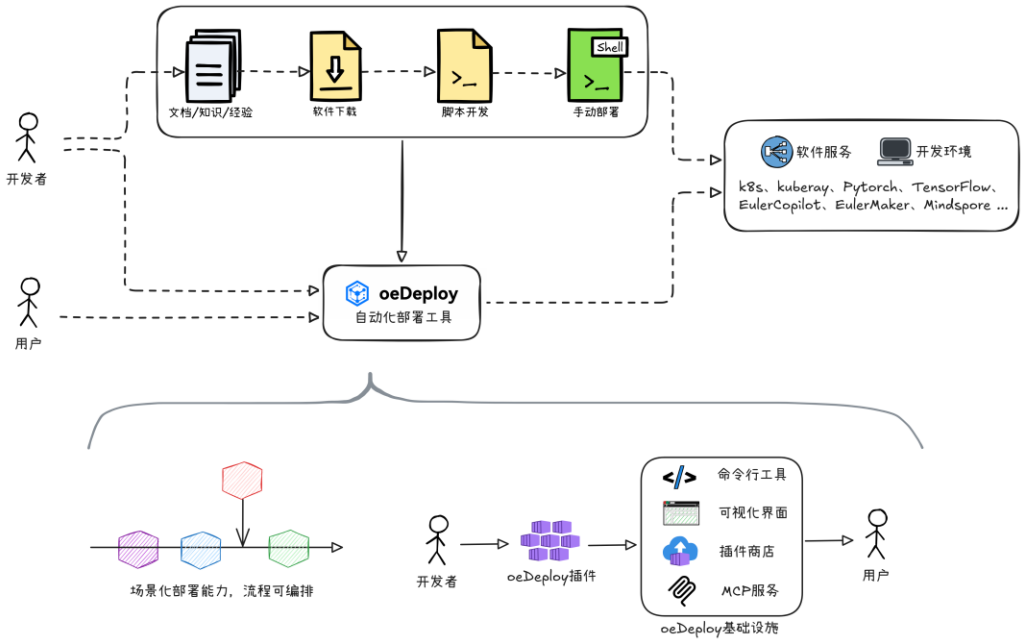
oeDeploy 是一款轻量级的软件部署工具，旨在帮助开发者快速、高效地完成各类软件环境部署，对单节点与分布式场景均可适配。

功能描述

多场景支持 & 主流软件一键部署：支持单节点应用与集群软件环境的一键部署，已经支持 openEuler 社区工具链、主流云原生软件、常用 AI 开发组件、常用 RAG 工具的快速部署，新增支持了 CANN 开发套件与昇腾 NPU 驱动的快速部署能力。

灵活的插件化管理 & 优秀的部署体验：oeDeploy 提供可扩展的插件架构，灵活管理多种部署能力，开发者也可以快速发布自定义部署插件。新版本 oeDeploy 优化了插件源的管理，支持一键生成本地插件源，便于开发者快速发布自定义插件。oeDeploy 支持极简的命令行操作方式，在软件商店 DevStore 上线，支持可视化操作。用更少的代码，实现更高效的软件部署体验。

高效部署 & 智能开发：oeDeploy 发布了 MCP 服务，在 DevStation 中实现开箱即用，借助大模型的推理能力，支持用自然语言完成各类软件的一键部署，部署效率提升 2 倍；支持将用户文档快速转换成可以直接运行的 oeDeploy 插件，开发效率提升 5 倍。



应用场景

ISV 与开发团队可以将 oeDeploy 作为向客户交付软件产品的统一形式。借助 oeDeploy 提供的命令行工具与插件框架，开发团队只需较少的开发工作量就可以实现优秀的部署效果，降低用户的额外学习成本，提高客户满意度。

面向开发者与维护人员，oeDeploy 提供了复杂软件环境的快速部署能力，可以在几分钟内部主流的 AI 训推框架，大幅降低软件开发门槛，避免了重复操作。同时，开发者可以基于 oeDeploy 发布自定义的部署能力，让更多用户受益于一键部署的便利。借助大模型与 MCP 的能力，oeDeploy 可以让部署流程更加高效，开发过程更加智能。

DevStation 开发者软件商店（DevStore）

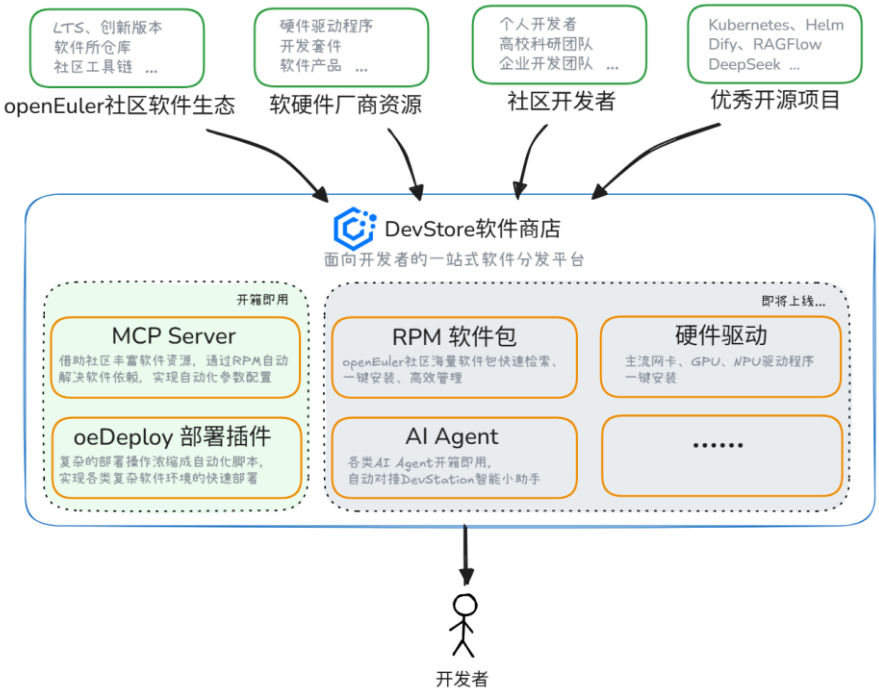
DevStore 是 openEuler 桌面版本的应用商店，是面向开发者的软件分发平台，支持 MCP 服务、oeDeploy 插件的检索与快捷部署功能。在 DevStation 平台上实现开箱即用。

功能描述

MCP 服务一键安装：DevStore 借助 openEuler 社区丰富的软件生态，以 rpm 软件包的形式处理 MCP 运行所需的软件依赖，并通过内置的服务管理工具，在智能体应用中快速部署 MCP 服务。自动帮助用户解决软件依赖与 MCP 配置问题，大幅提升用户体验。目前已支持 80+MCP 服务。

oeDeploy 插件快速部署：DevStore 借助 oeDeploy 工具实现主流软件的快速部署，大幅度降低开发者部署软件的时间成本。包括 Kubernetes、Kuberay、Pytorch、TenserFlow、DeepSeek 等 AI 软件，EulerMaker、DevStation 智能助手等社区工具链，以及 RagFlow、Dify、AnythingLLM 等主流的 RAG 工具。

在后续的版本中，DevStore 会继续吸纳上游社区、软硬件厂商的各类资源，陆续上线对 rpm、Agent、驱动等更多种类型软件的管理功能，支持检索与一键安装部署，进一步降低开发者的软件获取成本。



应用场景

DevStore 作为 openEuler 桌面端的软件商店，帮助开发者与初学者快速获取主流开发工具、AI 软件、MCP 服务等等，在详情页提供了丰富的用户文档与操作入口。所有复杂软件的部署操作都可以在一个操作界面完成，大幅降低学习成本、提升用户体验。

8. 特性增强

LLVM for openEuler 编译器

LLVM for openEuler 编译器基于开源 LLVM 软件进行深度打造，是面向服务器互联网行业、数据中心新应用、通算 AI 新场景和视频编解码等高算力场景的高性能编译器。同时在 openEuler 社区打造国内的 LLVM 基线，提供高性能、安全可靠、易创新的 LLVM 下游社区

稳定的发行版，已支持主流的系统语言（C/C++）和芯片架构（X86/AArch64/RISCV/LoongArch 等）。

功能描述

LLVM for openEuler 编译器在 openEuler 24.03 LTS SP3 版本引入以下编译特性，优化数据库、压缩解压缩等相关应用的运行效率，释放软件的极致性能。

AArch64 字节序转换加速优化：在常见数据库如 MySQL 等应用中，数据是以大端字节序进行存储，在加载到内存中使用的时候，需要先转换为小端字节序然后进行后续计算，这通常需要由"按 Byte 进行 load"-"移位"-"按位或"步骤循环操作，带来性能开销。该优化高效使能 AArch64 REV 系列指令，加大 load 位宽以减少 load 次数以及减少移位和按位或操作，提升应用性能。

Machine Sink 增强优化：在编译器优化过程中，通常会把指针类型的循环迭代变量的迭代操作（如循环内"ptr++"操作）下沉到循环末尾，如果循环下一迭代开始存在 load 操作并依赖该指针变化后的值时，就会出现数据依赖而造成指令 stall。该优化通过合理调度指令位置来避免因数据依赖而造成的指令 stall。

AArch64 Loop Idiom 优化：循环俚语优化用于识别广泛应用中常见的循环格式，使用 AArch64 SVE 等灵活的指令能力将其转换为优化版本的循环实现，来大幅提升循环的执行效率。

应用场景

通过 LLVM for openEuler 编译器的编译优化能力，能够帮助提升数据库、压缩解压缩场景中主流应用（如 MySQL、LZ4 等）的性能。

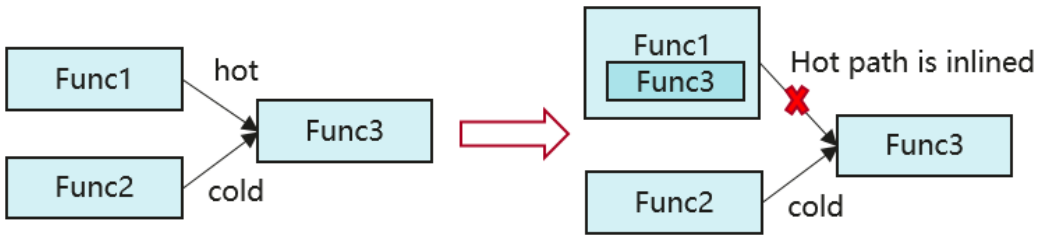
Go for openEuler 编译器

Go for openEuler 是基于开源 Golang 开发，是一款高性能、高可靠、易开发的 Golang 发行版，致力于打造生态兼容、极致体验、亲和 openEuler 的高性能编译器。主要面向云原生、微服务应用等对敏捷开发和运行性能都有要求的容器云场景，围绕业界主流 Go 业务负载进行优化，解决实际业务中由原生 Golang 能力不足导致的性能问题，并适配国产龙芯、鲲鹏等硬件平台，充分释放国产硬件算力。

功能描述

功能描述 1：CFGO 反馈优化

在保证程序功能不变的前提下，通过收集程序运行时信息，指导编译优化进行更准确的优化决策，获得性能更优的目标程序。基于程序局部性原理，使热指令紧密排布，优化 cache/TLB 命中，有效降低程序前端瓶颈，提升程序性能。



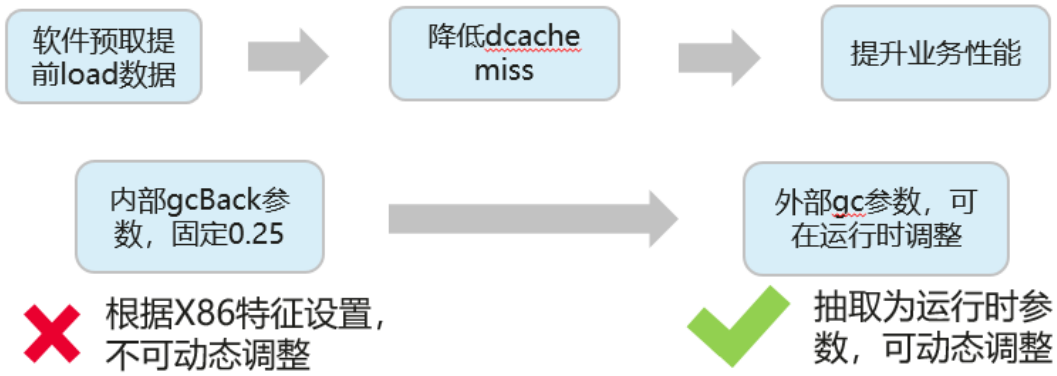
函数内联

功能描述 2：ARM 原子指令优化

在部分业务场景中，Golang 运行时调用 CAS 锁、LD/ST 指令开销较大，改为 ARM 亲合指令序列实现，可实现性能提升。

功能描述 3：运行时 GC 优化

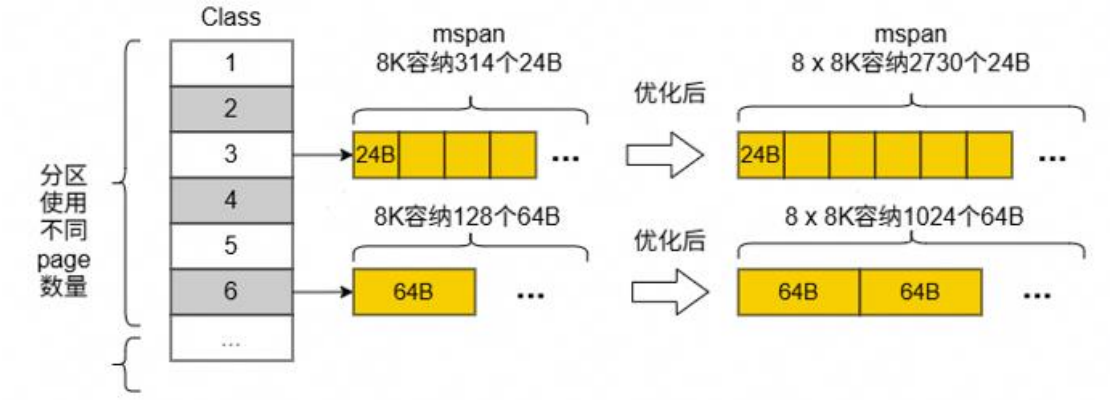
结合特征，插入软件预取；抽取 GC 协程开销资源参数为运行时参数，支持根据不同业务特征进行动态调整。



运行时 GC 优化

功能描述 4：内存页分区扩展优化

Go 默认使用 8KB 页为基础单位，在小对象固定使用单个 8KB 页管理时，容易出现频繁创建和查找内存对象的情况，通过对 classsize 对象分区以最优粒度页面进行内存分配和释放，可以有效提升内存管理效率，在 Go Malloc Benchmark 上性能平均提升 10%，其中 MallocLargeStruct 用例提升 50.29%。



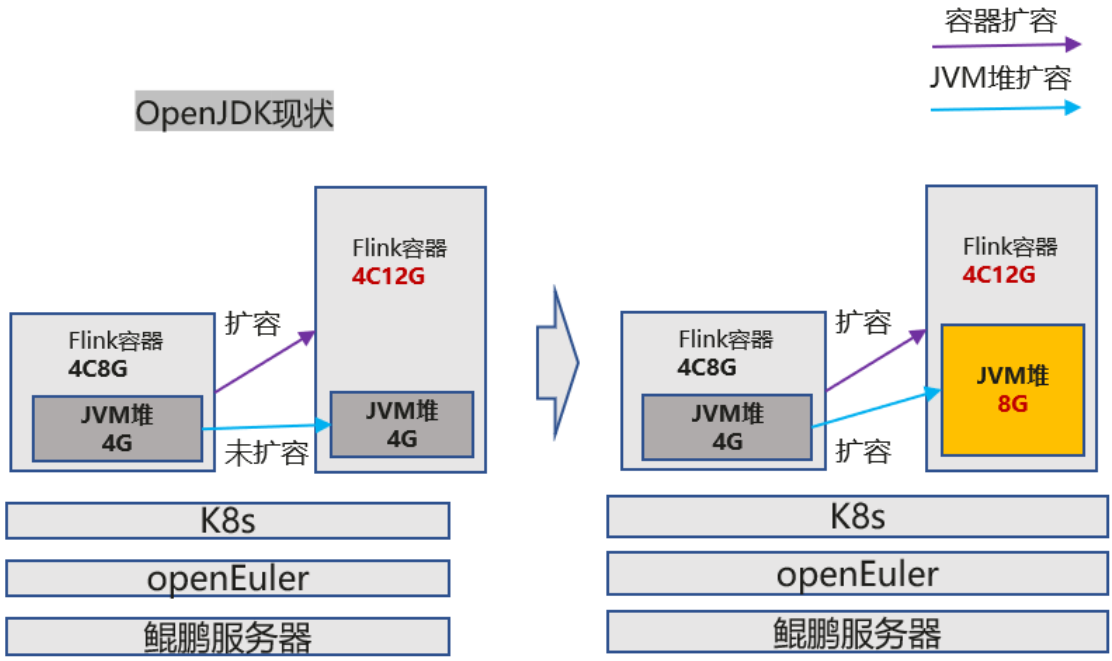
应用场景

Go for openEuler 是基于 Golang 开发的 Go 基础软件，在 openEuler 等 Linux 环境里应用广泛，应用场景涵盖云原生、分布式存储、云游戏等场景。

毕昇 JDK 支持堆内存扩容

功能描述

互联网容器化部署应用的模式下，大部分客户容器场景下容器资源支持垂直伸缩，当前 OpenJDK8 的最大堆只能在启动时支持修改，无法支持在线动态扩缩，java 应用无法在线使用到容器扩容出的内存，需要 java 应用启动时重新设置最大堆；鉴于此问题，毕昇 JDK 在毕昇 JDK8、毕昇 JDK21、毕昇融合 JDK 中的 G1GC、PSGC 上实现堆内存上限在线伸缩能力，允许用户在应用运行时动态更新 Java 堆内存的上限，而无需重启 JVM。



应用场景

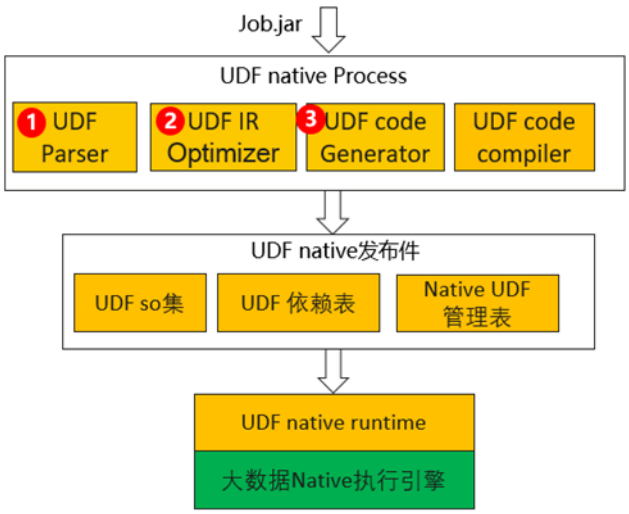
互联网等容器场景业务，在容器在线扩容后需要 java 业务的堆内存大小也支持在线扩容的场景。目前已落地京东大数据场景。

编译器 UDF 自动 native 框架

功能描述

针对开源大数据 JVM 执行效率低的缺点，UDF 自动 native 框架负责将 Java UDF 自动转换为 C/C++ Native UDF，并进一步从内存高效管理、硬件亲和等维度提升大数据处理性能。UDF 自动 native 框架致力于实现用户无感知、全自动的 Java UDF native 加速机制。UDF 自

动 native 框架主要由 UDF parser、UDF IR Optimizer、UDF code Generator、UDF code compiler 等模块组成：



UDF parser 将业务 jar 包字节码自动转换为 IR 代码，并基于 UDF 特征提取出 UDF 代码；UDF IR Optimizer 从内存对象自动管理、硬件亲和加速等维度对 UDF IR 进行优化；UDF Code Generator 将 UDF IR 对等转换为 native 代码；UDF code compiler 将 UDF native 代码在线编译为 native 二进制。最后，UDF 的 native 二进制发布到大数据执行节点上，由大数据系统 native 执行引擎动态加载执行，提升大数据系统处理性能。

应用场景

当前版本的 UDF 自动 native 框架适用于对接 Flink 大数据 native 执行引擎，通过配套 Flink Datastream 场景 native 基础库，可以在用户无感知的情况下实现 Flink Datastream UDF 自动 native 加速。

毕昇 JDK17 支持退优化可观测

JDK17 的 JFR Streaming API 功能，是 JFR 从“事后静态分析”迈向“实时监控”的关键特性。

在传统的 JFR 使用模式中，流程是：记录 -> 停止记录 -> 转储为 .jfr 文件 -> 用 JMC 离线分析。这种模式是“事后分析”，对于排查已经发生的问题非常有效。

Streaming API 引入了一种全新的模式：它允许 Java 应用程序在不中断 JFR 记录、不生成完整 .jfr 文件的情况下，实时、持续地从 JVM 内部订阅和消费 JFR 事件流。

功能描述

在使用 Streaming API 的时候，通过本功能可以在 ****** 处获取当前时间之前的一段 jfr event，例如退优化事件。

```
// 1. 创建一个 RecordingStream
RecordingStream rs = new RecordingStream();
// 2. 启用我们感兴趣的事件并配置设置
rs.enable("jdk.GCPhasePause").withPeriod(Duration.ofSeconds(1));
rs.enable("jdk.Deoptimization").withPeriod(Duration.ofSeconds(1));
// 3. 订阅特定事件并设置事件处理器（回调函数）
rs.onEvent("jdk.GCPhasePause", event -> {
// 从事件中读取字段
Duration duration = event.getDuration("duration");
String name = event.getString("name"); // 例如 "GC Pause"
    **********
    shell:    jcmd JFR.start delay=-1 filename=xxx.jfr
    **********
});
// 4. 启动流（这是一个非阻塞调用）
rs.startAsync();
```

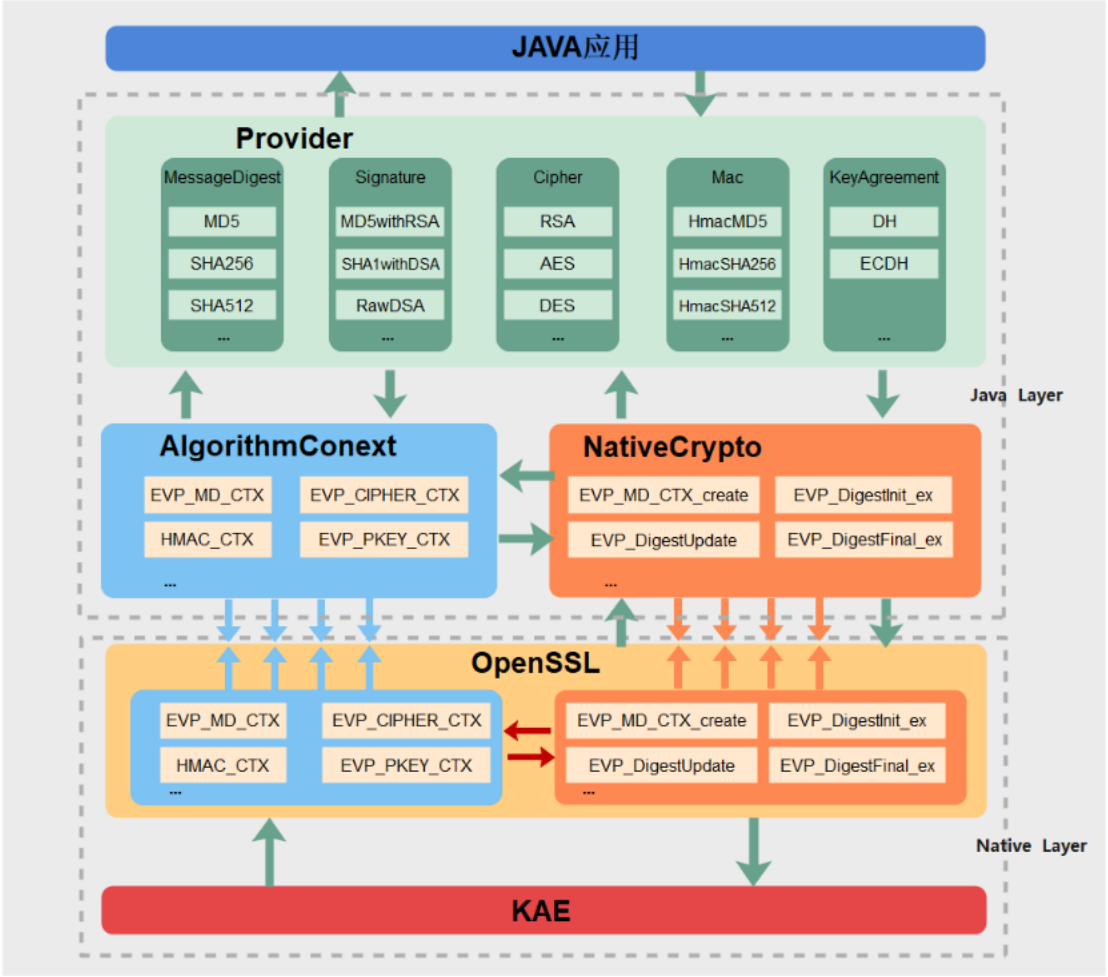
应用场景

在线监控和事后分析配合的模式，在事后分析的 jfr 文件中，能够包含当前时间之前的一段时间内的所有 jfr 事件。

毕昇 JDK21 KAE 特性增强

KAE 加解密是鲲鹏加速引擎的加解密模块，其依托修改 OpenSSL 库实现对底层 KAE 硬件的调用。

因此应用代码只需调用 OpenSSL 库，KAE 作为 OpenSSL 的底层引擎提供加速能力。KAEProvider 介于应用和 OpenSSL 之间，提供一种便利的使用 OpenSSL 库及使能 KAE 加速的方式。



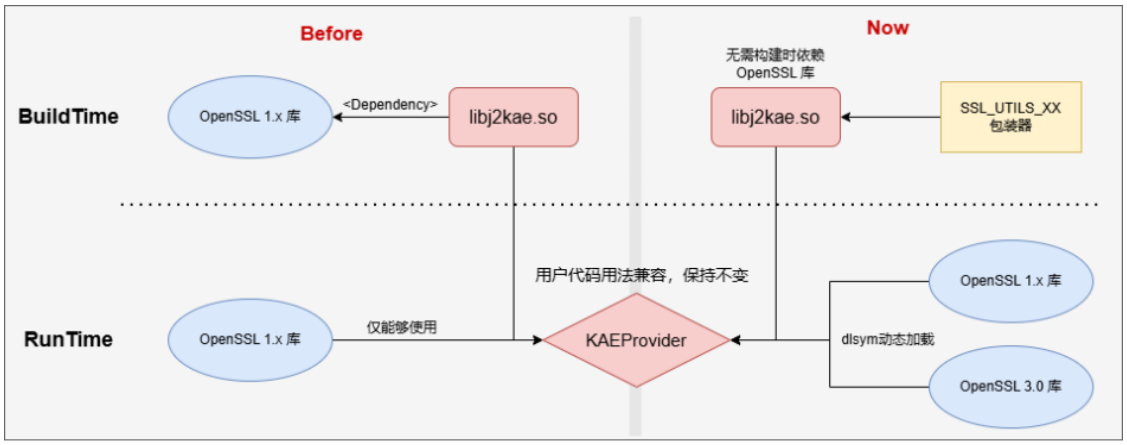
KAE 压缩模块支持 ZLIB、GZIP 数据格式压缩，压缩带宽大大提升。现 Java 应用中或多或少都会使用到压缩解压缩特性，尤其在一些数据存储、http 等场景压缩 CPU 占比可能高达 70%+，开源 zlib 压缩速度成为瓶颈，若 Java 应用可以使能 KAE-zlib 将极大提高业务的性能。

功能描述

过去的版本里 KAEProvider 支持 OpenSSL1.x 版本。随着 2021 年 OpenSSL3.0 版本发布，在架构上、可扩展性、安全性上都有所提升，是一个重大版本更新，后续进行安全通信开发的用户很可能会选择采用升级版本。然而 OpenSSL1.x 到 OpenSSL3.0 一部分 API 发生了废弃、新增或变化，因此 KAEProvider 无法直接支持使用，需要进行代码适配。

本特性添加后 KAEProvider 构建结果不变，但构建过程中不再依赖构建环境本地的 OpenSSL 库环境(之前构建目标产物 libj2kae.so 时需依赖构建环境下的 OpenSSL1.x 环境)，使用哪个 OpenSSL 版本会在程序运行时动态判断和加载。

KAEProvider 用法保持不变，并且用户使用本增强特性时也需要手动打开（修改 OPENSSL_ENGINES 环境变量和 kae.useOpenSSLVersion 配置），不修改原有加解密逻辑，不影响 JDK 原有功能和使用方式。



对于 KAE zip 支持，JDK 中将动态库拆分：拆分 libzip.so 为 libzip.so 和 libz.so，使其分别包含 JNI 和 zlib 代码。使能 KAE-zlib：通过-DGZIP_USE_KAE 参数来控制使能，压缩解压缩过程 Java 侧不做文件头尾的处理，使其压缩数据结构的生成和解析均依赖于底层的 zlib 库的 ZLIB、GZIP 压缩算法来完成。

应用场景

涉及加解密、压缩解压缩的场景，KAE Provider、KAE zip 对于支持的算法能够有效提升性能。

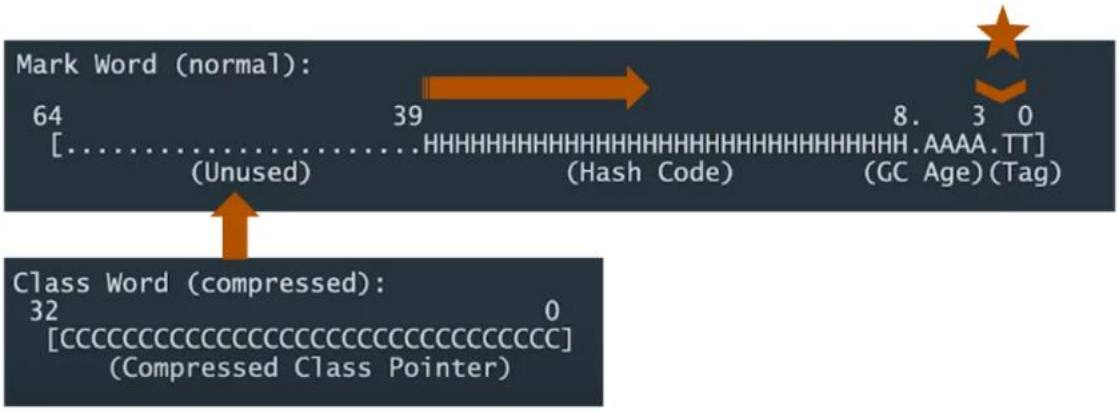
毕昇 JDK21 元数据压缩

Java 对象的存储会有额外的开销，即对象头，它用于存储与该对象相关的元数据信息。伴随存活对象的增多，以及大量小对象的存在，对象头占用空间问题也会愈发严重。本特性在 Aarch64 平台下将对象头从 128/96 bits 降到 64bits。

功能描述

在 64 位 Hotspot 中，Java 对象具有 128 位的对象头：一个 64 位的 MarkWord 字和一个 64 位的类指针。对象的平均大小通常为 5-6 个字，其中两个字始终由对象头占用。该特性利用了正常情况下 MarkWord 高位未被使用的特点，将类指针移动到 MarkWord 的高位，并通过缩短 HashCode 和修改标志位等方式，将对象头压缩至 64 位。

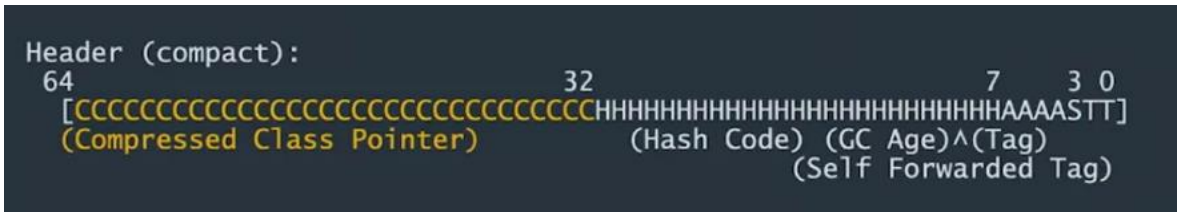
修改前的对象头：



该特性针对对象头做了以下修改:

- 将类指针部分移到 MarkWord 的高位，类指针仅支持压缩类指针（基本可以满足绝大数应用）。
- HashCode 减少到 25 位，这对于超大 Hashtable 的性能可能会存在劣化。
- 增加了 1 位 Self Forwarded Tag，用来表示 GC 复制阶段失败的情况（原始是将 MarkWord 替换成指向自己的指针）。

修改后对象头如下:



因此我们就可以显著降低内存压力，实现:

- 减少堆使用量
- 更高的对象分配率
- 降低 GC 活性
- 对象更紧密地排布将带来更好的缓存局部性

应用场景

本方案适用场景较为通用，对于内存占用的减少根据应用程序的实际数据情况而变化。本特性针对小对象分配多的场景下能够在端到端体现出较好的效果，在大数据部分场景下有收益。

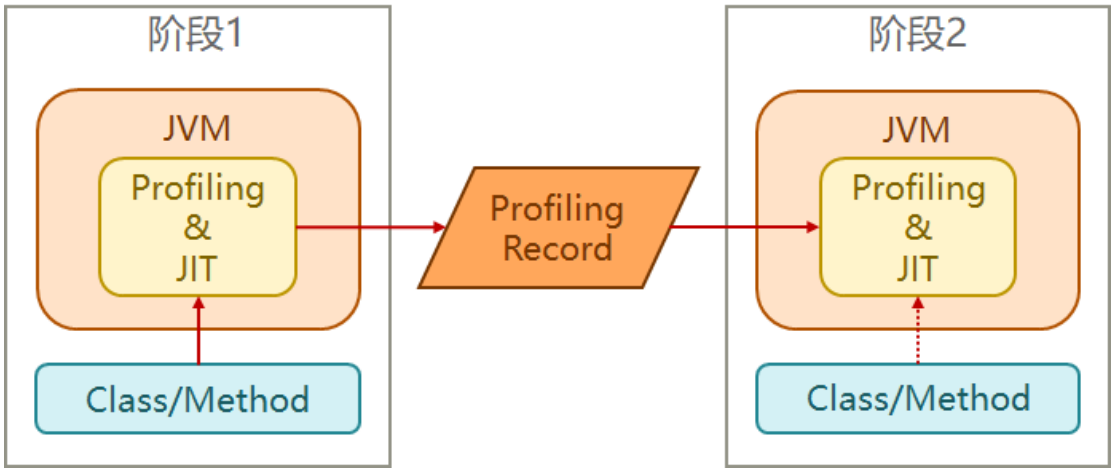
毕昇 JDK21 JIT 预热增强

本特性旨在使能 Java 进程启动后快速抵达峰值性能。Java 进程从拉起到最终到达峰值，存在一个较为漫长的过程。其中解释器->采样->编译 C1->进一步采样->编译 C2 的采样与在线 JIT 编译流程是应用预热过程的重要一环。本特性通过压缩该部分流程，有效加速 VM 启动与预热，助力应用快速达峰。

功能描述

毕昇 JDK JProfilecache 方案将应用程序的发布分成了两个阶段，分别是：

- **记录阶段**：在程序运行结束时，把热点方法的 Profiling 信息(主要是方法调用次数，回边次数)及该方法所属类(包括父类)输出到指定文件中。
- **预编译**：后续启动的时候，JVM 读取热点方法所在类并进行预加载，同时会把热点方法加入到编译队列进行提前编译，使得在用户请求到来之前，跨过 Profiling 阶段，就把热点方法编译成为性能较高的 Native Code，减少热点方法的预热开销。



应用场景

适用于对启动速度有要求的云原生场景、涉及反复拉起 driver/executor 进程的 Spark 大数据启动敏感场景。本特性在此类场景下能够有效加速 VM 启动与应用预热，实现性能快速达峰。

毕昇 JDK8 特性增强

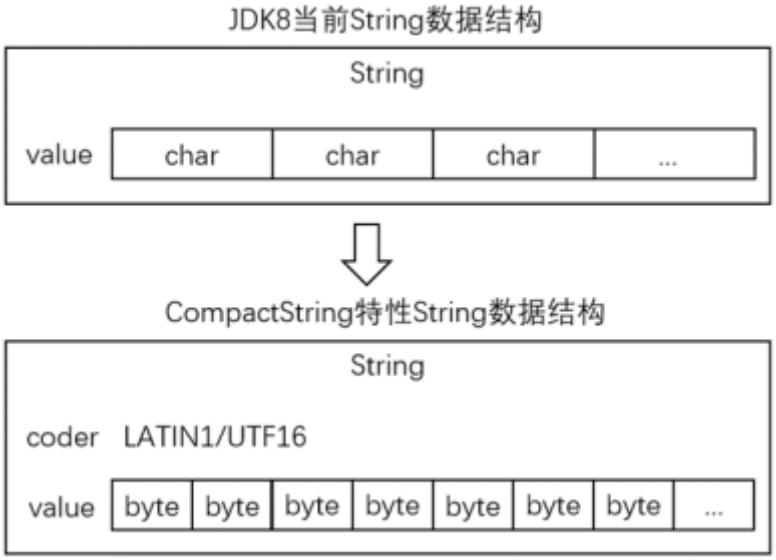
CompactStrings（紧凑字符串）是 Java 9 引入的一项重要内存优化特性，用于改进字符串的存储方式，显著减少内存占用。在 Java 8 及之前版本中，字符串内部使用 char[]字符数

组存储，每个字符占用 2 个字节（UTF-16 编码），无论字符串实际内容如何。这导致在处理大量仅包含 ASCII 字符的字符串时，内存使用效率低下。目前，CompactStrings 已从 JDK17 回合到毕昇 JDK8，进一步提升毕昇 JDK8 整体性能。

功能描述

CompactStrings 将 String 对象内部的表示由一个 UTF-16 的 char 数组改为一个 byte 数组加上一个 encoding 标识字段。这个新的 String 类可以根据字符串中的内容将字符编码为 Latin-1（一个字符 1byte）或者 UTF-16（一个字符 2byte），encoding 标识用于指示这个 String 是如何编码的，具体如下：

- 对于仅包含 Latin-1 字符的字符串，使用 byte[]字节数组存储，每个字符仅占用 1 字节。
- 对于包含非 Latin-1 字符（如中文、日文等）的字符串，仍然使用 char[]存储，每个字符占用 2 字节。
- 每个字符串对象包含一个编码标识（coder）字段，用于指示当前字符串的编码方式：
    - LATIN1（值为 0）：只包含 Latin-1 字符。
    - UTF16（值为 1）：包含其它字符（如中文、日文等）。



应用场景

CompactStrings 可以节省字符串内存占用，对于仅包含 Latin-1 的字符串，内存占用节省约 50%。因此，CompactStrings 适用于字符串密集型应用中，可以极大节省内存占用，并进一步减少 GC 频率，提升整体性能。

GCC for openEuler CFGO 反馈优化特性增强

日益膨胀的代码体积导致当前处理器前端瓶颈成为普遍问题，影响程序运行性能。编译器反馈优化技术可以有效解决此类问题。

CFGO（Continuous Feature Guided Optimization）是 GCC for openEuler 的反馈优化技术名，指多模态（源代码、二进制）、全生命周期（编译、链接、链接后、运行时、OS、库）的持续反馈优化，主要包括以下两类优化技术：

- 代码布局优化：通过基本块重排、函数重排、冷热分区等技术，优化目标程序的二进制布局，提升 i-cache 和 i-TLB 命中率。
- 高级编译器优化：内联、循环展开、向量化、间接调用等提升编译优化技术受益于反馈信息，能够使编译器执行更精确的优化决策。

功能描述

GCC CFGO 反馈优化共包含三个子特性：CFGO-PGO、CFGO-CSPGO、CFGO-BOLT，通过依次使能这些特性可以缓解处理前端瓶颈，提升程序运行时性能。为了进一步提升优化效果，建议 CFGO 系列优化与链接时优化搭配使用，即在 CFGO-PGO、CFGO-CSPGO 优化过程中增加-flto=auto 编译选项。

- CFGO-PGO

CFGO-PGO 在传统 PGO 优化的基础上，利用 AI4C 对部分优化遍进行增强，主要包括 inline、常量传播、去虚化等优化，从而进一步提升性能。

- CFGO-CSPGO

PGO 的 profile 对上下文不敏感，可能导致次优的优化效果。通过在 PGO 后增加一次 CFGO-CSPGO 插桩优化流程，收集 inline 后的程序运行信息，从而为代码布局和寄存器优化等编译器优化遍提供更准确的执行信息，实现性能进一步提升。

- CFGO-BOLT

CFGO-BOLT 在基线版本的基础上，新增 aarch64 架构软件插桩、inline 优化支持等优化，进一步提升性能。

应用场景

CFGO 通用性较好，适用于整机性能瓶颈在 CPU 上，且 CPU 瓶颈在前端的 C/C++应用，如数据库、分布式存储等场景，一般可以取得 5~10%性能提升。

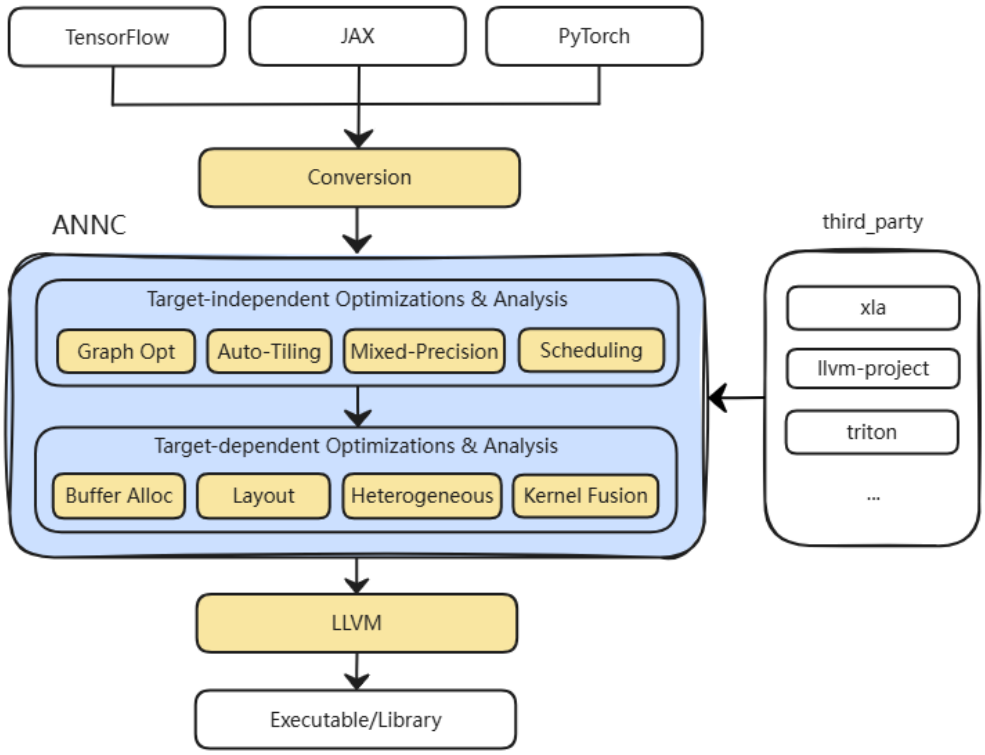
AI 编译器

ANNC（Accelerated Neural Network Compiler）是专注于加速神经网络计算的 AI 编译器，聚焦于通过计算图优化，高性能融合算子生成和高效的代码生成和优化能力，加速推荐和大语言等模型的推理性能，并且从架构设计角度支持业界主流开源推理框架和不同硬件后端的接入，提高软件的可扩展性。

功能描述

计算图优化是通过优化神经网络的计算流程，从算法角度减少冗余操作、混合精度改写和自动子图调度优化计算负载和提高缓存利用率，从硬件架构角度优化张量数据布局、算子融合和转换、子图切分调度，进一步优化负载，充分利用硬件资源。

高性能融合算子库生成和对接包括前端计算图模式识别和转换，高性能算子库查询和对接，以及算子库自动生成三部分，从汇编指令层面上通过数据预取、模型并行和新型指令集应用等优化手段，减少访存，提高并行计算效率。



ANNC 旨在通过图编译优化和高性能算子生成和对接，提高 AI 推理速度和降低功耗，达到提高用户单位成本的推理效率的目的，同时通过软件兼容性和易用性设计减少用户的运营成本和环境影响。

应用场景

当前版本的 AI 编译器主要聚焦于图编译优化和算子库选择调用，面向主流搜索推荐系统中的粗排、精排模型能够取得较大收益，尤其是面向特征处理复杂的嵌入层网络，图编译优化能够取得更好的效果。

同时，ANNC 也预留了大语言模型等未来场景的性能优化途径，对接 LLM 推理框架，结合现有的图算融合、内存布局优化技术，以及 GEMM 等核心的性能优化手段，减小推理时延，提高推理吞吐。

CCA 机密计算

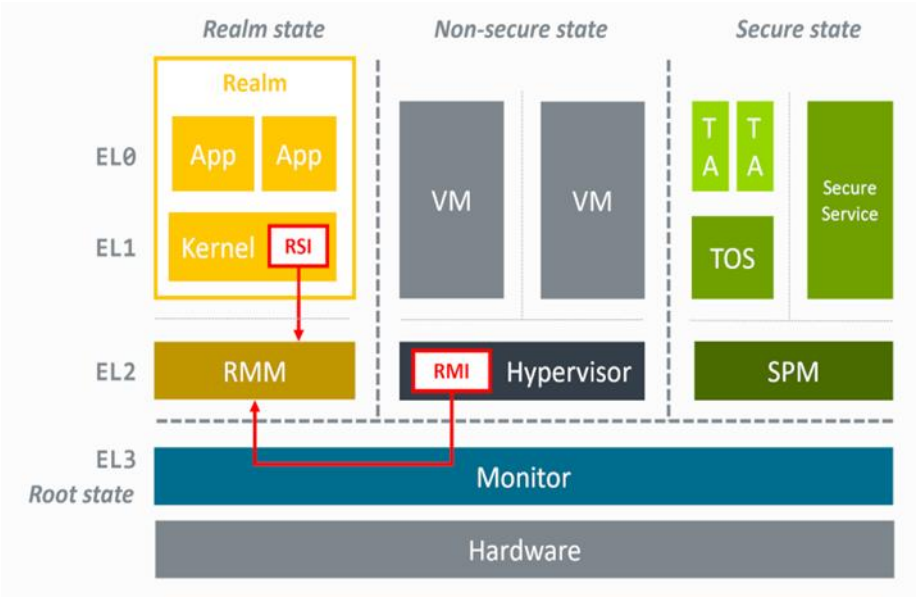
ARM CCA（Confidential Computing Architecture）是 ARM v9 新引入的机密计算架构规范，旨在为下一代计算设备定义标准化的机密计算解决方案。CCA 引入了全新的 Realm 域

作为可信执行环境，来保护正在使用中的数据和代码的机密性和完整性，即使面对拥有特权的基础设施软件或云服务提供商，也能得到有效保护。

openEuler 基于 ARM CCA 机密计算架构规范，实现了 OS 相关组件（KVM、QEMU、libvirt、Guest kernel）对 CCA 的支持，提供了原生支持 Realm 机密虚机的社区版本，满足基于 Realm 机密虚机保护使用中数据的安全诉求，同时提供了兼容传统应用生态及虚机管理软件的易用性。

功能描述

ARM CCA 通过以下核心组件协同工作，构建一种隔离的、受保护的执行空间，在代码执行和数据访问方面与正常世界完全隔离，成为 Realm 机密域。



- **Realm 机密域**：Realm 是 CCA 的核心抽象，它是一种与正常世界（Non-secure）和安全世界（Secure，原来的 Trustzone）并行的新类型执行环境。Realm 是硬件隔离的，专为托管敏感代码和数据而设计。它独立于主机操作系统和 Hypervisor，它们可以管理 Realm 但无法访问其内部内容。
- **动态管理**：Hypervisor 可以应客户要求动态创建 Realm，并为其分配内存和 CPU 资源。但在 Realm 初始化后，Hypervisor 会将其控制权移交给一个受保护的安全虚拟化模块 RMM（Realm Management Monitor），此后 Hypervisor 便无法访问 Realm 内的秘密。

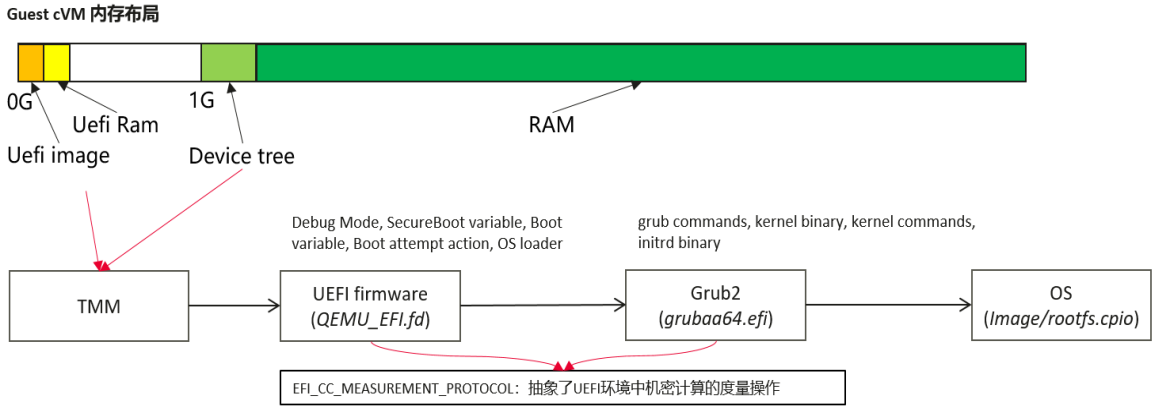
- **内存管理**：CCA 扩展了系统内存管理单元（MMU），使其能够识别和隔离 Realm 内存。任何从 Realm 外部（包括 Hypervisor）发起的访问尝试都会被硬件阻断，从而确保数据的机密性。
- **远程证明**：每个支持 CCA 的处理器都有一个基于硬件的唯一身份标识。当 Realm 启动时，它可以生成一份由硬件密码学签名的证明报告（Attestation Token）。用户可以获得这份报告，并验证其签名及组件度量值，从而确信他们的工作负载正在一个真实的、未被篡改的 ARM CCA 环境中运行。

应用场景

机密虚机主要应用于云计算环境，为核心工作负载提供高级别安全隔离。它使得客户能在不受信的公有云上处理敏感数据（如金融记录、医疗健康信息或知识产权），并确保其机密性和完整性连云服务商也无法窥探。这有效满足了数据主权、监管合规和跨机构隐私数据协作（如联合建模与分析）的关键需求。

virtCCA 特性增强

功能描述



当前 virtCCA 架构在启动方式上存在特定约束：其仅支持 **kernel 与 rootfs 分离的启动模式**（即内核镜像与根文件系统分别挂载）。然而，在主流云平台环境中，虚拟机的启动流程普遍依赖 **GRUB 引导机制**，这要求将 UEFI 固件（如 EDK2）、内核（Kernel）及初始内存文件系统（initramfs）整合至单一磁盘镜像（如 QCOW2 格式）中。功能要点包括：

1. 单镜像封装

(1) 将 EDK2 固件、GRUB 引导程序、内核（Kernel）及 initramfs 整合至单一 QCOW2 磁盘镜像，形成完整启动栈。

(2) GRUB 通过配置文件（grub.cfg）定位内核路径，要求内核与 initramfs 必须位于同一文件系统（如 EXT4/XFS）。

2. 安全信任链传递

(1) Secure Boot 机制：EDK2 验证 GRUB 及内核的数字签名，确保启动组件未被篡改。

(2) 硬件资源协同：依赖 UEFI 运行时服务枚举硬件设备，为虚拟机管理程序（如 KVM）提供虚拟化资源池

3. 云原生优化

(1) 支持快照克隆、根文件系统动态扩容（依赖 initramfs 中的 cloud-init 工具）特性。

应用场景

云环境中采用 UEFI（Unified Extensible Firmware Interface）启动模式，已成为现代虚拟化架构的核心技术，virtCCA 架构支持 UEFI 启动，扩展了机密虚机的应用场景：

1. 快速实例启动与弹性伸缩

UEFI 通过**并行硬件初始化**（如 CPU、内存、存储设备同步检测）显著缩短启动时间。

2. 大容量磁盘支持

UEFI 依赖 **GPT 分区表**，突破传统 MBR 的 2TB 磁盘限制，支持**百 TB 级云盘**（如阿里云 ESSD 云盘），满足大数据存储（如 HDFS）、AI 训练等场景需求。

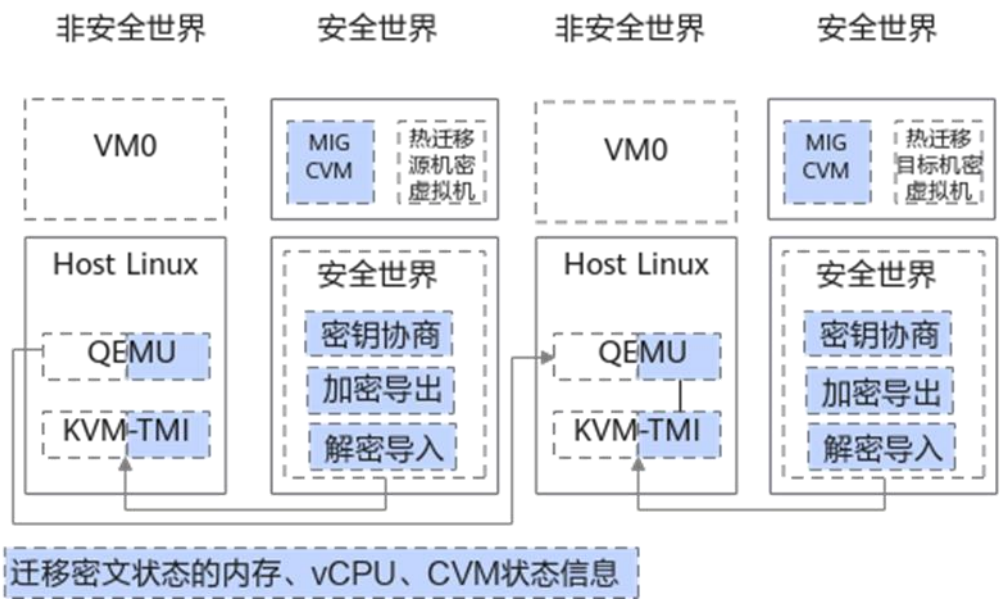
3. 自动化运维与批量部署

镜像标准化：云服务商提供的 UEFI 兼容镜像默认启用 GPT 分区，简化用户部署流程。

virtCCA 机密虚机热迁移

功能描述

机密虚拟机热迁移是指在机密虚机（CVM）业务运行不中断的情况下，将其从一个机密计算环境安全地迁移至另一个经过验证的机密计算环境，并确保机密虚机中的敏感数据在迁移前、中、后始终处于加密或隔离保护之下。



virtCCA 机密云主机热迁移通过迁移管理虚机 MigCVM 和安全世界 Hypervisor TMM 组件实现热迁移过程中机密虚机状态的机密性和完整性，包括如下功能要点：

- 迁移源平台需要对目标平台进行身份验证，TMM 组件需要对 MigCVM 进行可信度量；
- 身份验证通过后，源平台和目标平台 MigCVM 完成迁移密钥协商；
- 迁移源平台和目标平台建立加密安全会话，确保迁移数据的安全；
- MigCVM 对迁移状态进行管理和跟踪，确保迁移过程状态安全，管理异常状态；
- TMM 组件负责导出、导入 virtCCA 机密虚机内存状态及 VCPU 状态；
- TMM 组件基于迁移密钥对机密虚机状态数据进行加密，并对状态数据进行完整性校验。

应用场景

主流云主机管理平台要求云主机支持热迁移能力，通过热迁移能力实现在服务器运维场景下将云主机从一台服务器迁移至另一台服务器，热迁移过程中保证客户业务的连续性。VirtCCA 机密云主机热迁移主要包括以下应用场景：

- 硬件、系统维护与升级

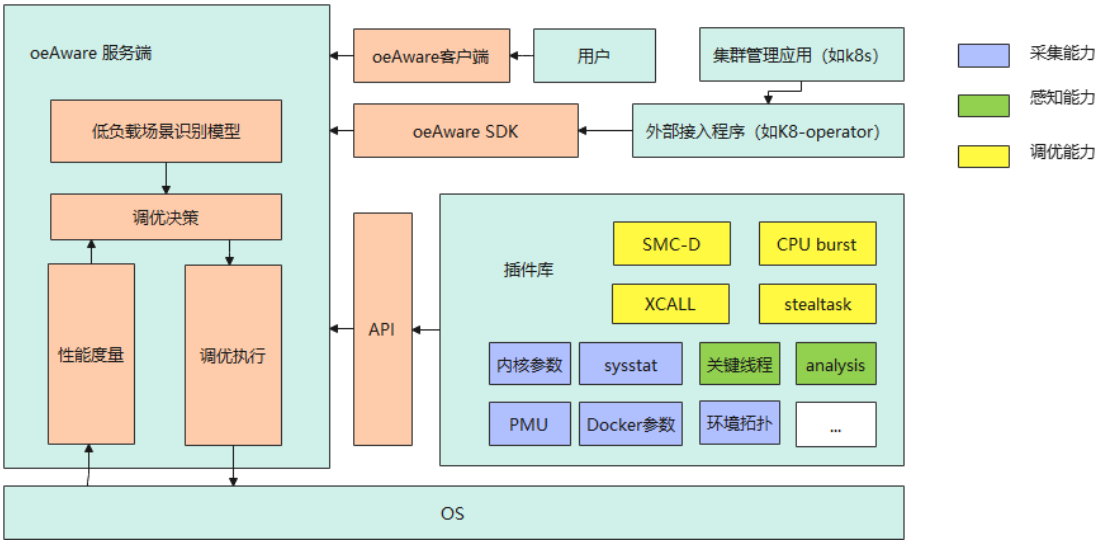
  物理服务器进行固件升级、硬件更换或系统更新时，可将 CVM 虚拟机实时并安全地迁移到其他主机，业务零中断。

- 负载均衡与性能优化

当某台主机 CPU、内存负载过高时，可以将指定资源消耗较大的 CVM 虚拟机到负载较低的主机，灵活调度，确保性能整体最优或局部最优。

oeAware 采集、调优插件等功能增强

oeAware 是在 openEuler 上实现低负载采集感知调优的框架，目标是动态感知系统行为后智能使能系统的调优特性。传统调优特性都以独立运行且静态打开关闭为主，oeAware 将调优拆分为采集、感知和调优三层，每层通过订阅方式关联，各层采用插件式开发尽可能复用。



功能描述

oeAware 的每个插件都是按 oeAware 标准接口开发的动态库，包含若干个实例，每个实例可以是一个独立的采集、感知或调优功能集，每个实例包含若干个 topic，其中 topic 主要用于提供采集或者感知的数据结果，这些数据结果可供其他插件或者外部应用进行调优或分析。

- SDK 提供的接口可以实现订阅插件的 topic，回调函数接收 oeAware 的数据，外部应用可以通过 SDK 开发定制化功能，例如完成集群各节点信息采集，分析本节点业务特征。
- PMU 信息采集插件：采集系统 PMU 性能记录。
- Docker 信息采集插件：采集当前环境 Docker 的一些参数信息。
- 系统信息采集插件：采集当前环境的内核参数、线程信息和一些资源信息（CPU、内存、IO、网络）等。

- 线程感知插件：感知关键线程信息。
- 评估插件：分析业务运行时系统的 NUMA 和网络信息，给用户推荐使用的调优方式。
- 系统调优插件：（1）stealtask ：优化 CPU 调优 （2）smc_tune（SMC-D）：基于内核共享内存通信特性，提高网络吞吐，降低时延 （3）xcall_tune ：跳过非关键流程的代码路径，优化 SYSCALL 的处理底噪。
- Docker 调优插件：利用 cpuburst 特性在突发负载下环境 CPU 性能瓶颈。

约束限制

- SMC-D：需要在服务端客户端建链前，完成使能 smc 加速。比较适用于长链接多的场景。
- Docker 调优：暂不适用于 K8s 容器场景。
- xcall_tune：内核配置选项 FAST_SYSCALL 打开。
- realtime_tune ：需要配合 Preempt-RT 内核使用。
- net_hard_irq_tune ：TCP 网络通信业务。

应用场景

stealtask 适用于希望提高 CPU 利用率的场景，例如 Doris，该调优实例可以有效提高 CPU 利用，避免 CPU 空转。

XCALL 适用于应用程序的 SYSCALL 开销较大的场景，XCALL 提供一套跳过非关键流程的代码路径，来优化 SYSCALL 的处理底噪，这些被跳过的非关键流程会牺牲部分维测和安全功能。

SMC-D 特别适用于需要高吞吐量和低延迟的应用场景，如高性能计算（HPC）、大数据处理和云计算平台。通过直接内存访问（DMA），SMC-D 能够显著减少 CPU 负载并提升交互式工作负载的速率。

cpuburst 适用于高负载容器场景，例如 Doris，能够缓解 CPU 限制带来的性能瓶颈。

realtime 适用于需要低时延、低抖动、有严格时延限制的场景，如工业制造行业。

net_hard_irq_tune 适用于业务性能受 TCP 网络影响较大的业务，例如 redis、nginx 等。

并行感知调度适用于跨 NUMA 访存对业务性能比较大且不好固定绑核的场景，例如 Spark。

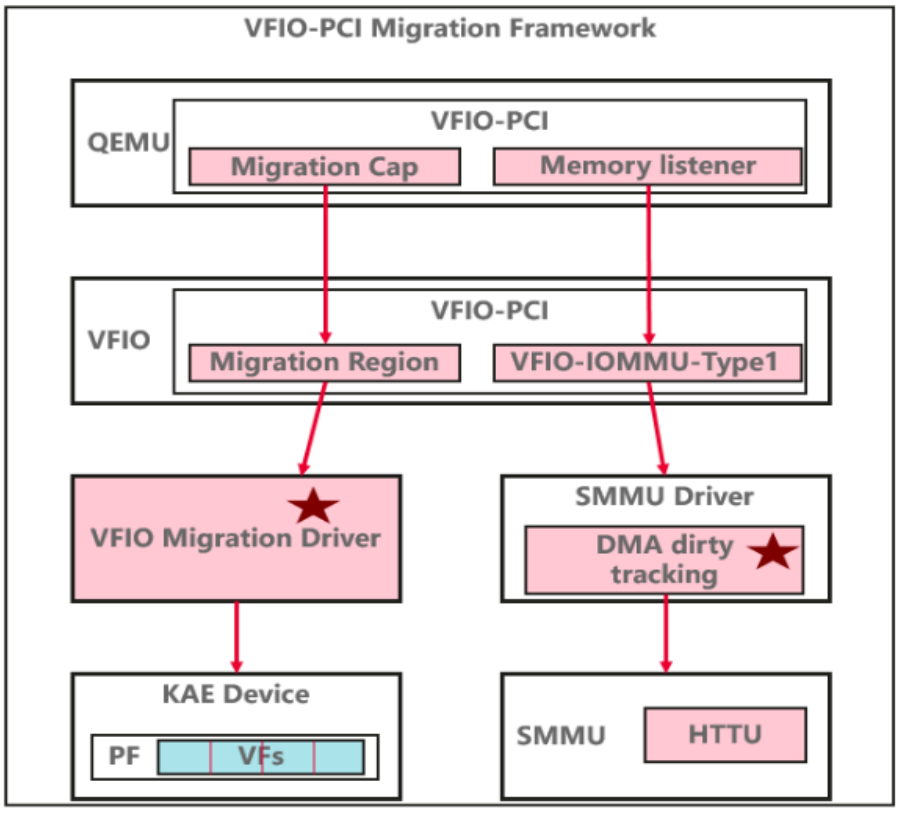
容器潮汐调度适用于容器化部署的业务，例如容器化 mysql，当业务负载低时，在满足 QoS 的基础上，使用最少的 CPU 资源，减少 CPU 迁移、idle 切换、cache miss 和跨 NUMA 访问，增强资源的局部性。当业务负载高时，又可以突破绑核约束，使用更多 CPU 资源来提升 QoS 质量，提高 CPU 资源利用率。

虚拟化支持 vKAE 直通设备热迁移

功能描述

KAE 是基于鲲鹏 920 新型号处理器提供的硬件加速解决方案，包括 HPRE、SEC、ZIP 设备，可用于加解密和压缩解压缩，能够显著降低处理器消耗，提高处理器效率。KAE 直通热迁移是指虚拟机在配置 KAE 直通设备时，进行热迁移的能力，可以为 KAE 设备的使用提供更强的灵活性和业务不中断的保障。

smmu 脏页跟踪是实现直通设备高效、可靠的热迁移的关键技术。在 ARM 架构中，通过纯软件方式进行脏页跟踪，会带来了较大的性能损耗。HTTU(Hardware Translation Table Udate)允许硬件自动更新 smmu 页表状态，在进行写操作时会自动置对应页表项的写权限位，热迁移时扫描页表的写权限位进行脏页统计。



应用场景

为虚拟机中使用 KAE 直通设备的场景提供热迁移支持，适用于对数据安全性和处理性能有较高要求的领域，如金融、云计算和大数据处理等，有效增强业务连续性与运行稳定性。

Global Trust Authority 远程证明

功能描述

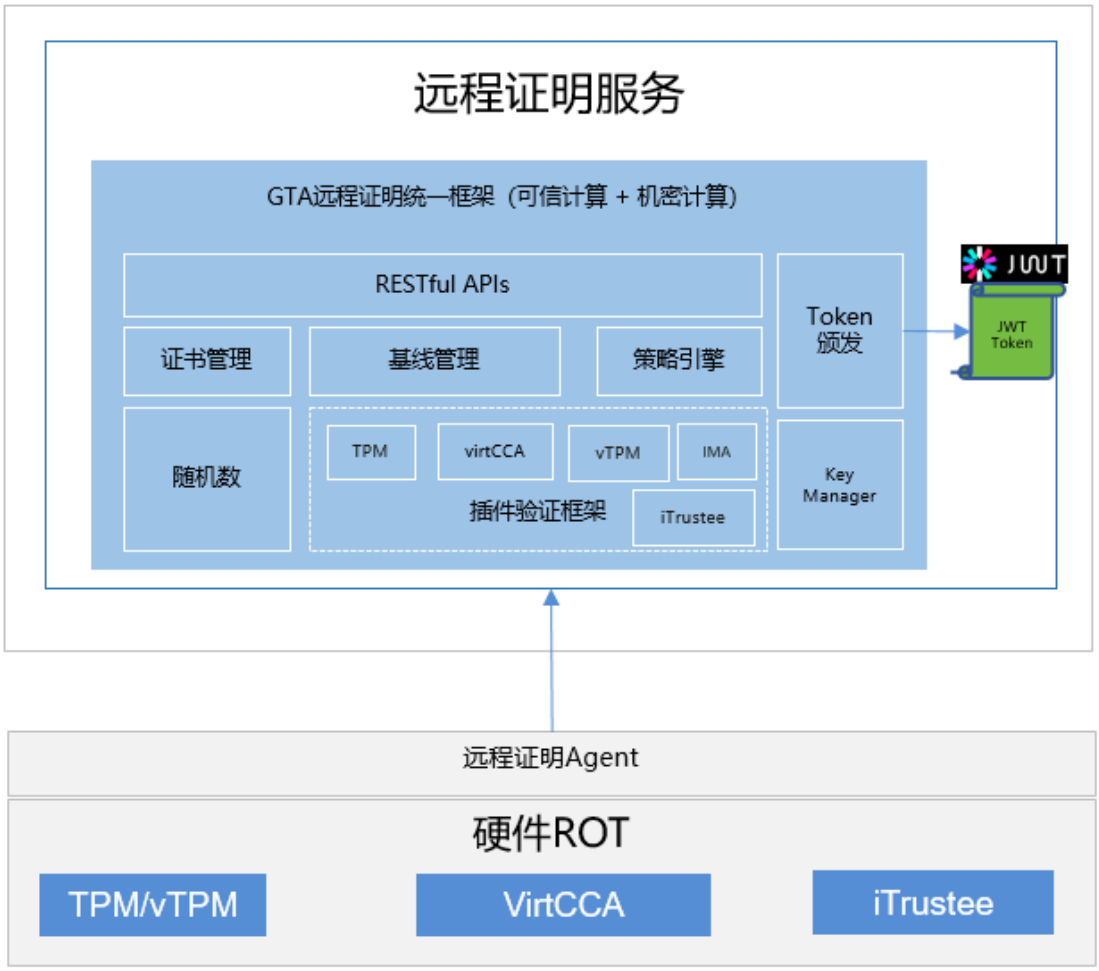
GTA 远程证明服务组件支持 TPM/vTPM、VirtCCA 及其 IMA 的远程证明，分为客户端和服务端。

- 服务端提供了远程证明服务框架兼容可信计算及机密计算，支持证书、策略等的增删改查，Quote 验证，随机数，JWT Token 生成等能力。
- 客户端支持采集本地 TPM 证据，并可与服务端交互，验证 Quote。

本组件在安全性及易用性上也提供了多种能力。

安全性上支持数据库完整性保护、数据链路加密，验证防重放，SQL 防注入，用户隔离，密钥轮换机制等一系列差异化安全竞争力。

易用性上支持护照模式及背调模式。客户端支持定时上报，响应挑战等多种验证模式。客户端及服务端支持 rpm 包及 docker 安装部署。



应用场景

远程证明是开启机密计算及可信计算的前置条件。只有当运行环境在密码学意义上被严格证明安全可信后方可进行后续运算。所有涉及机密计算的端到端解决方案都应将远程证明作为整体解决方案中的核心一环，一旦运行环境被证明不可信，则应立即中止后续任务。在 AI 模型保护，用户隐私保护，密钥管理等多场景均有广泛应用。

Kuasar 机密容器

功能描述

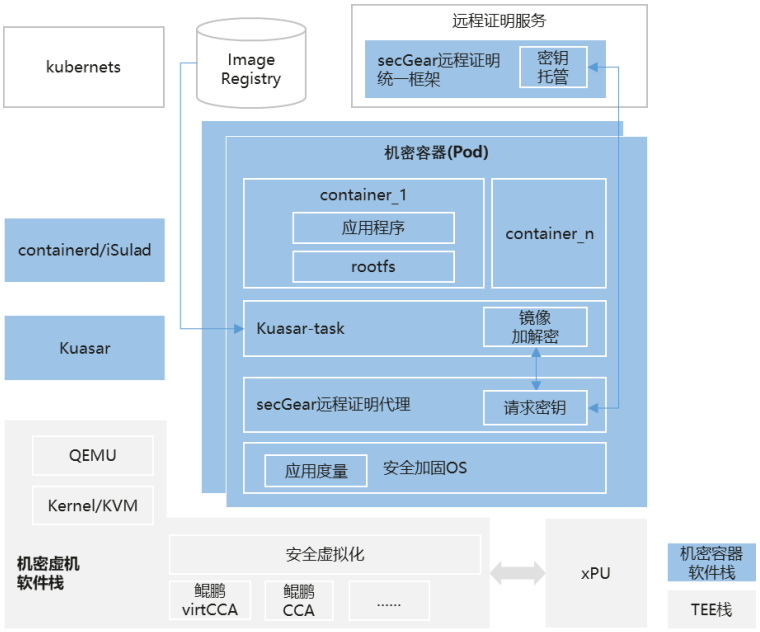
Kuasar 统一容器运行时在支持安全容器的基础上添加了对机密容器的支持。用户可以通过配置 iSulad 的运行时参数，完成对 Kuasar 机密容器的纳管。

当前 Kuasar 机密容器使用 iSulad+Kuasar 方案，提高了启动速度，极大降低了内存底噪。一方面 Sandbox API 的实现，使得创建容器不再单独创建 pause 容器，节省了准备

pause 容器镜像快照的时间；另一方面得益于 1:N 的管理模型，Sandboxer 进程常驻，从而节省了冷启动 Shim 进程的时间，这使得容器的启动速度大大提升，带来与 Pod 数成正比的内存收益。最后，Kuasar 使用 rust 实现，相比 golang，内存更安全，语言本身也带来了一些内存收益。

支持功能特性:

- 支持 iSulad 容器引擎对接 Kuasar 机密容器运行时，兼容 Kubernetes 云原生生态。
- 支持基于 virtCCA 的机密硬件，允许用户在鲲鹏 virtCCA 可信执行环境中部署机密容器。
- 支持 secGear 远程证明统一框架，遵循 RFC9334 RATS 标准架构，允许在机密计算环境中运行的容器向外部的受信任服务证明其可信性。
- 支持在机密容器内部拉取并解密容器镜像，保护容器镜像的机密性和完整性。

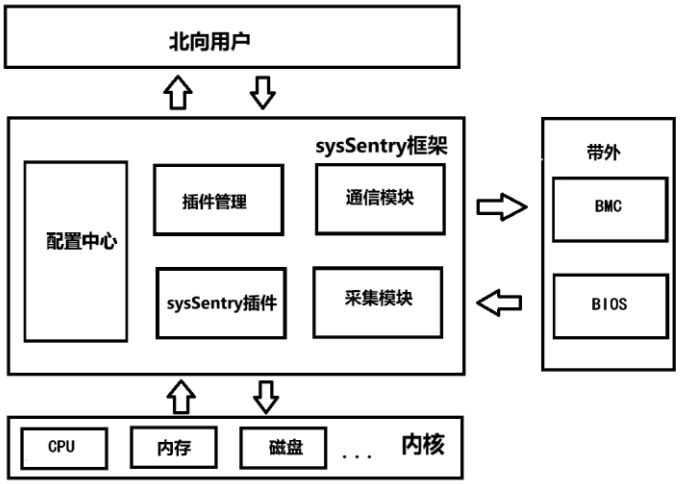


应用场景

Kuasar 机密容器在满足客户数据安全的诉求下，兼容云原生生态，确保用户的机密应用具备高可用性、弹性伸缩、快速交付 等云原生优势，在 AI 保护、数据可信流通、隐私保护等机密计算的业务中有广泛的应用场景。

sysSentry 系统级故障管理框架

sysSentry 主要提供故障巡检框架，该框架通过提供统一的北向故障上报接口以及南向提供支持不同巡检/诊断能力的插件，支持对系统中 CPU、内存、磁盘、NPU 等硬件故障进行巡检和诊断。



sysSentry 功能设计如下：

统一告警/事件通知服务：通过提供一个统一的告警服务，接收各个插件上报的故障信息，并由该通知服务进行统一转发，各个业务订阅服务可以根据需要进行不同故障的消息订阅。

统一日志服务：通过提供统一的日志服务，支持各个插件的故障信息进行汇总记录，提升问题定位效率。

故障诊断/巡检框架：该框架支持以插件化的方式进行各项巡检任务以及诊断任务的开发和配置，不同插件支持独立启动、停止、状态查询、结果查询以及启动方式设置，并且支持 C/C++、Python、Shell 等不同编程语言的插件。

轻量级数据采集服务：该服务支持通过内核接口、BIOS、BMC 等接口，查询硬件的各个状态信息，供各个插件进行分析和使用，并且支持适配底层不同的架构、版本以及数据采集服务。

慢 IO/慢盘检测

功能描述

慢 IO 检测基于滑动窗口对系统中一段周期内不同硬盘的 io 时延数据进行分析，当整个窗口中异常周期的数量超过一定数量时，则认为此时该盘发生慢 io 事件。

目前支持两种类型的慢 io 检测插件：基于平均时延计算异常阈值的平均阈值插件 avg_block_io，和基于 AI 算法聚类计算阈值的 AI 阈值插件 ai_block_io；两种插件均支持最多 10 个 io 阶段：blk-throttle、wbt、iocost、get_tag、plug、deadline、bfq、kyber、hctx、driver。

目前支持检测的硬盘类型有 NVMe SSD，SATA SSD，SATA HDD。

应用场景

1. 硬件故障：由于硬盘损坏、老化原因等导致硬盘出现慢 IO 情况。
2. IO 栈异常：由于内核 IO 栈配置、处理异常等导致出现慢 IO 情况。
3. 业务高负载：由于业务负载导致出现 IO 资源使用超限的情况。

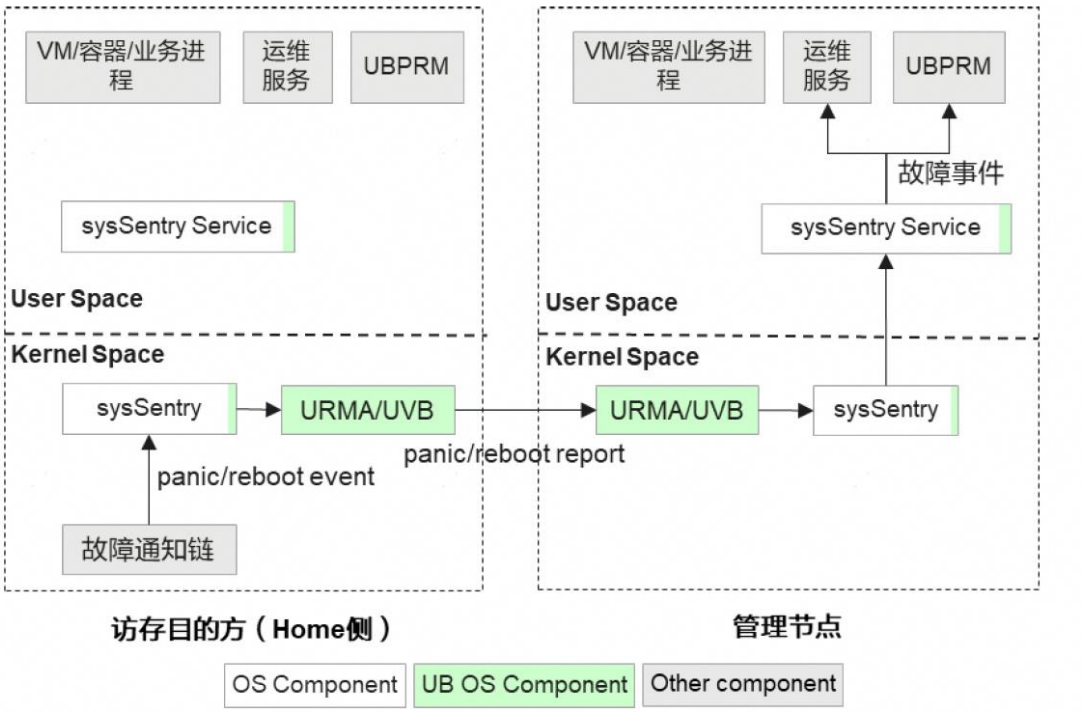
灵衢可靠性插件

功能描述

当紧急事件（OOM、Panic、reboot 等）发生时将相关的紧急事件阻塞，并上报事件到 UBPRM 中，防止发生数据丢失或业务中断，同时也负责统一管理通过 UB event 上报的事件。

节点故障检测与恢复

当 Home 侧节点发生 Panic/reboot 时，该节点借出的内存将无法访问，会影响 User 侧的业务进程，在此情况下需要将保存在 Home 侧内存中的数据迁移出来。



详细通知流程如下：

Home 侧节点内核故障流程中（例如 panic、reboot、shutdown 流程等）通知 sysSentry 即将复位重启。

1. sysSentry 通过 URMA/UVB 通道进行故障事件广播。
2. 故障事件从 Home 侧节点的 URMA/UVB 通道发送到管理节点的 URMA/UVB 通道。
3. 管理节点的 sysSentry 收到故障事件，通过 sysSentry Service 通知到管理节点 UBPRM。
4. UBPRM 可通过 sentryctl 相关命令配置 sysSentry 开启对 Panic/reboot 事件劫持、配置跨节点通信必备参数配置，通过订阅 OS Panic 以及 OS reboot 类型事件来监听 Panic/reboot 事件。

当 UBPRM 收到 sysSentry Service 上报的故障通知后，可采取相应的故障隔离和恢复措施，例如将 Home 侧节点内存迁移至其他节点并解除和故障节点的借用关系，确保使用池化内存的业务不受影响。

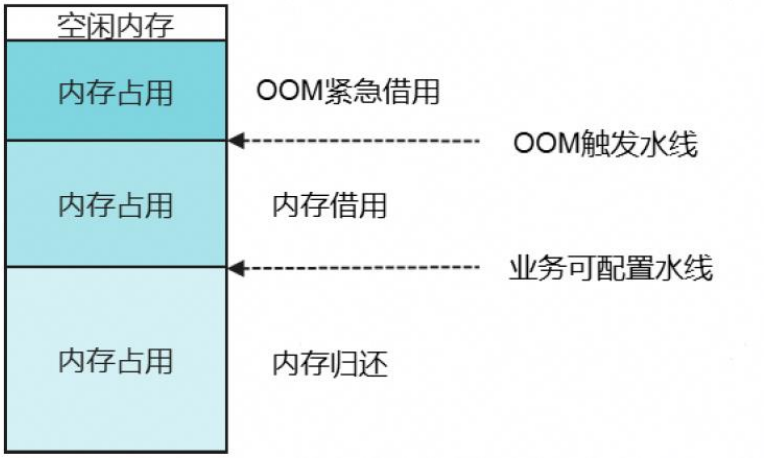
处理完成后，需返回故障处理结果，系统按照如下流程进行恢复：

1. UBPRM 通知 sysSentry Service 故障处理结果。
2. 故障处理结果经过 sysSentry 和 URMA/UVB 通道发送到 Home 侧节点 URMA/UVB 通道。
3. Home 侧节点 sysSentry 从 URMA/UVB 通道拿到故障处理结果。
4. Home 侧节点返回内核故障流程，继续复位重启。

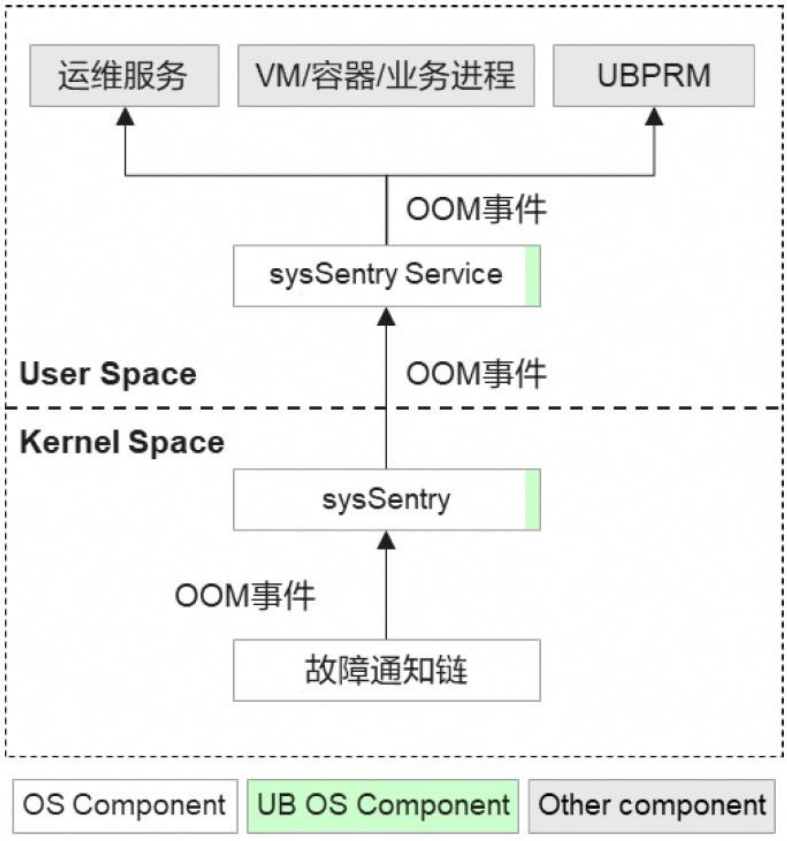
5. Home 侧节点如果在一定时间内未收到 UBPRM 的通知，则同样会复位重启。

OOM 检测、预防与恢复

在内存借用场景下，UBPRM 会按照固定的时间周期检测内存占用水位情况，该水位可配置，根据实际的使用情况采用内存归还/内存借用的策略。当短时间内发生大量的内存占用，导致内存水位迅速上涨时，UBPRM 检测机制可能无法及时发现该现象，导致业务节点发生进程被杀死或节点重启，发生业务中断。此时可通过 OOM 预防机制将 OOM 事件上报给 UBPRM，触发 OOM 的紧急借用策略。总体处理策略如下图所示：



当触发 OOM 紧急借用时，通知流程如下：



1. 发生 OOM 的节点通过 OOM 故障通知链上报 OOM 事件至 sysSentry。
2. sysSentry 将 OOM 事件上报至 sysSentry Service。
3. sysSentry Service 将 OOM 事件上报至 UBPRM 或运维服务。

UBPRM 可订阅 sysSentry 的 OOM 类型事件，当节点发生 OOM 事件后，UBPRM 会收到 sysSentry 服务上报的事件通知，并做相应的处理，建议处理流程：

1. UBPRM 借用其他节点内存，并上线到业务节点的 OS。
2. OS 在 OOM 中尝试申请内存，如果申请到，则返回给对应的应用，否则杀死对应的应用。如果是内核态申请内存出现 OOM，则走入 Panic 流程整机复位。

应用场景

灵衢内存池化场景下，节点发生异常故障导致借用内存即将失效时，通过劫持对应故障事件，通知管控面进行内存紧急迁移，避免业务中断。

支持树莓派

Raspberry Pi（树莓派）是由 Raspberry Pi 基金会与 Broadcom 公司合作开发的一系列小型单板计算机。凭借其价格低、体积小、能耗低、高可编程性以及丰富的生态系统等特点，树莓派在工业自动化、机器人技术、物联网、教育以及业余爱好者项目等领域得到了广泛应用。树莓派 4B 和树莓派 5 作为树莓派产品线的经典代表，采用 ARM 架构的处理器。其中树莓派 4B 是极具性价比的普及型单板计算机，树莓派 5 凭借其显著的性能突破和扩展能力成为一款在高性能边缘计算领域颇具竞争力的创新产品。

功能描述

作为开源硬件领域的一个较为高阶的硬件产品，树莓派 4B 和树莓派 5 支持 Raspberry Pi OS、Ubuntu、openEuler 等多种 Linux 发行版，外设丰富，具有较强的视频编解码能力，以及板载网络等功能，完全可以作为独立计算机系统使用。

应用场景

树莓派 4B 和树莓派 5 凭借其强大的性能和丰富的扩展能力，广泛应用于多个领域：

1. 教育与学习：学习 Python 等编程语言、借助外设接口进行电子实验；
2. 多媒体与娱乐：作为媒体中心或游戏机；

3. 物联网和智能家居：作为传感器节点或智能家居中枢，用于环境监测、家庭自动化控制和边缘计算；
4. 服务器与网络应用：用于家庭服务器、轻量级 Web 服务及容器化应用；
5. 创客与 DIY 项目：用于机器人控制、3D 打印管理和无人机飞控；
6. 科研与开发：用于 AI 实验、嵌入式开发原型验证；
7. 工业与自动化：实现设备监控、人机界面和机器视觉。

RISC-V 架构特性增强

ISA-L

ISA-L，全称英特尔智能存储加速库（Intel® Intelligent Storage Acceleration Library），是一个专门针对存储应用优化的开源函数库，其中包含 EC，CRC，RAID 以及压缩解压的优化计算代码。

功能描述

ISA-L，全称英特尔智能存储加速库（Intel® Intelligent Storage Acceleration Library），是一个专门针对存储应用优化的开源函数库，其中包含 EC，CRC，RAID 以及压缩解压的优化计算代码。ISA-L 初始的加速代码只支持 x86 架构，其他架构均使用基础的 C 语言代码进行计算，并无加速效果。在 ISA-L 库迭代的过程中，逐渐支持了 ARM 架构的汇编加速代码，但对于 RISC-V 架构的支持大部分还停留在基础的 C 语言通用实现。这也致使 ISA-L 加速库在 RISC-V 平台上无法完全发挥其完整硬件特性的能力，存在着巨大的性能提升空间。

为提升 RISC-V 平台上 ISA-L 加速库的计算能力，我们主要做了以下优化：

（1）CRC 算法优化

核心瓶颈：CRC 算法主要涉及模除运算，但由于除法运算比较耗时，所以通用的实现方式是使用预先计算的查表法来加速这个计算。但是查表法的处理稍显繁琐且数据吞吐量较低，大部分架构使用汇编代码编写的多项式折叠算法来加速计算，数据吞吐量极高。目前 ISA-L 在 RISC-V 平台上还依赖基础的查表法实现，缺乏对应 RISC-V 架构的优化，因此存在性能瓶颈。

优化方案：

1. 专用汇编实现：依据 RISC-V 指令集特点，实现多项式折叠算法和 Barrett 缩减的汇编优化，用以替换基础查表法计算。

2. 寄存器优化分配：合理运用 RISC-V 通用寄存器，减少内存访问，将每次计算结果存储在寄存器中，降低访问延迟。
3. 指令流水线调度：结合 RISC-V 处理器流水线特征，通过指令重排与循环展开，充分利用指令执行单元，提升指令级并行度与吞吐率。

应用场景

ISA-L 存储加速库的 RISC-V 架构 CRC 计算性能优化主要应用于各类数据校验的场景，为保证数据正确性和完整性的校验提供加速支持。它使得 RISC-V 的设备可以用较快的速度完成压缩和解压数据的校验任务，有效解决了 RISC-V 生态系统在 CRC 计算中的性能瓶颈问题。

CRC 算法优化应用场景

关键应用领域：

1. 压缩数据流：数据压缩解压同时计算或验证 CRC 校验值，确保源数据正确性。能够在追求吞吐量的场景中哦，无缝集成数据校验。
2. 网络数据传输：ISCSI 协议在数据包层面继承了 CRC 校验，优化算法可以保障数据咋网络中的高速、可靠传输。
3. 防范静默数据损坏：行业广泛采用 T10-DIF 标准，核心为 CRC 校验，优化算法可以加速实现端到端保护，防止将错误数据当做正确数据使用。

Snappy 压缩算法

Snappy 是 Google 开发的一种快速数据压缩/解压缩库，旨在实现极高的速度和合理的压缩率之间的平衡。它的设计目标是速度，而不是最大限度地压缩数据。

功能描述

极速压缩与解压缩： Snappy 的核心优势在于其极快的速度 . 它主要使用字典编码（Dictionary Coding）和字面量（Literal）拷贝结合的方式进行数据压缩。在压缩过程中，Snappy 会在输入数据中查找重复的字节序列。如果发现重复序列，它将用一个引用（Reference）来替换该序列，这个引用指向该序列在先前已处理数据中的位置和长度。对于不重复的数据，则作为字面量直接保留。

低 CPU 开销： 由于设计上专注于速度，Snappy 的算法复杂度较低，对 CPU 资源的需求相对较小，非常适合需要在低延迟环境下快速处理大量数据的场景。

非面向最大压缩率： Snappy 的压缩率通常不如 gzip 或 bzip2，但其速度远高于这些算法。它适用于牺牲部分压缩率来换取处理速度的场景。

流式处理友好： Snappy 的数据格式设计使其易于进行流式（Streaming）压缩和解压缩。

RISC-V SIMD 指令集优化： 利用 RISC-V 向量扩展（Vector Extension, V-Extension）或打包 SIMD（Packed SIMD）指令，对 Snappy 压缩和解压缩过程中涉及到的字节查找（如查找重复序列）和批量数据拷贝操作进行向量化处理。这可以显著提高查找效率和数据吞吐量。

针对 RISC-V 内存模型优化： 优化内存访问模式，以更好地利用 RISC-V 架构的缓存层次结构，例如对字面量拷贝和引用跳转的数据访问进行对齐和预取，减少缓存缺失（Cache Misses）。

应用场景

Snappy 广泛应用于需要高速数据处理和存储的场景，例如：

大数据存储与处理系统： 在 HDFS、Spark、HBase 等大数据生态系统中，Snappy 常用于压缩存储的数据块和网络传输的数据流，以提高 I/O 效率并降低网络延迟。

日志文件压缩： 快速压缩大量的日志文件，既能节省存储空间，又不会因为压缩操作本身而成为系统瓶颈。

内存数据结构快照： 快速生成内存数据库或缓存系统快照，以实现持久化或容灾。

内部 RPC/网络通信： 快速压缩在服务之间传输的数据，减少网络带宽占用，同时保持低延迟。

高性能 RISC-V 服务器与集群： 在采用 RISC-V 架构的高性能计算（HPC）集群、数据中心服务器中，利用这些底层优化，可以进一步提升大数据工作负载的 I/O 性能和整体系统吞吐量。

边缘计算与嵌入式 RISC-V 设备： 在资源受限但仍需处理数据的边缘设备上，优化的 Snappy 库能以更低的功耗和更少的时钟周期完成数据压缩/解压缩，提高设备的能效比。

LZ4 压缩算法

LZ4 是一款高性能的实时数据压缩算法，广泛应用于操作系统内核、文件系统、网络传输等对性能要求严苛的场景。本技术白皮书详细阐述了 LZ4 在 RISC-V 架构上的非对齐内存

访问优化方案，通过硬件特性检测与针对性优化策略，显著提升了 RISC-V 平台上的压缩性能。

功能描述

非对齐内存访问优化机制

LZ4 算法在处理数据压缩时，频繁涉及非对齐内存访问操作。在不同 CPU 架构上，非对齐内存访问的性能表现差异显著。根据 ARM 平台的优化经验（参考 ARMv6 架构优化案例），当硬件原生支持高效非对齐内存访问时，直接内存访问策略相比传统的 memcpy()方法可带来较大性能提升。

本次优化针对 RISC-V 架构特性，特别是支持 Zicclsm 扩展的处理器平台，实现了智能化的内存访问策略选择机制：

1. 智能硬件检测：通过 GCC 编译器内置宏__riscv_zicclsm 精确检测 RISC-V 处理器是否支持 Zicclsm 扩展。
2. 分级优化策略：保留原有的三级优化策略体系（优先级 0 > 1 > 2），将支持 Zicclsm 扩展的 RISC-V 平台归类至最高性能的第 2 级策略。
3. 安全兼容设计：确保优化仅在硬件明确支持的环境下激活，对不支持 Zicclsm 扩展的 RISC-V 平台及其它架构保持原有行为。

应用场景

云原生与边缘计算环境

在基于 openEuler 的 RISC-V 云服务器和边缘计算设备中，LZ4 作为核心压缩组件，其性能直接影响系统整体效率。本优化特别适用于：

- 大规模数据处理：在大数据分析、日志处理等场景中，高频的数据压缩/解压缩操作受益于性能提升。
- 实时数据传输：网络通信、远程存储等对延迟敏感的应用场景，压缩性能提升直接改善用户体验。
- 资源受限设备：在 RISC-V 架构的 IoT 设备和嵌入式系统中，性能提升意味着更低的功耗和更长的电池续航。

AI 与高性能计算

在 AI 训练和推理工作负载中，数据预处理阶段常涉及大量数据压缩操作。优化后的 LZ4 在以下场景表现尤为突出：

- 模型参数压缩：加速 AI 模型在 RISC-V 平台上的部署和执行。
- 训练数据处理：提升数据加载和预处理速度，减少 GPU/CPU 等待时间。
- 分布式计算：在 RISC-V 集群环境中，优化节点间数据传输效率。

Openssl 加解密

OpenSSL 作为全球应用最广泛的密码学算法库，其核心算法性能直接决定了整个信息系统在数字签名、身份验证、数据加密与解密等关键安全操作中的执行效率。

功能描述

随着 RISC-V 开放指令集架构的快速发展和广泛应用，当前 OpenSSL 在 RISC-V 平台上仍主要依赖于 C 语言的通用实现，缺乏针对 RISC-V 架构特性的专用优化，这导致在 RISC-V 平台上的密码学算法性能远未达到硬件所能提供的理论极限，存在巨大的性能优化空间和迫切的技术改进需求。

为提升 RISC-V 平台上 OpenSSL 算法性能，我们主要做了以下优化：

（1）RSA 算法优化

核心瓶颈：RSA 算法性能主要由模乘、模幂运算的效率所决定，而 Montgomery 乘法作为优化模乘、模幂运算的常用手段，在其他主流架构上都存在高度优化的汇编优化代码。然而，随着 RISC-V 架构的快速发展和广泛应用，当前 OpenSSL 在 RISC-V 平台上仍仅依赖于 C 语言的通用实现，缺乏针对 RISC-V 架构特性的专用优化，因此存在性能瓶颈。

优化方案：

- 专用汇编实现：依据 RISC-V 指令集特点，实现 Montgomery 乘法的汇编优化，优化通用 C 代码的性能瓶颈。
- 寄存器优化分配：合理运用 RISC-V 通用寄存器，减少内存访问，将运算数据尽可能保留在寄存器中，降低访问延迟。
- 指令流水线调度：结合 RISC-V 处理器流水线特征，通过指令重排与循环展开，提升指令级并行度与吞吐率。

（2）AES-128-CBC 算法优化

核心瓶颈：AES-128-CBC 算法的性能主要受限于分组加密运算效率和矩阵变换操作，C 语言通用实现缺少 RISC-V 架构矢量加密扩展指令集加速。多块解密操作可并行化，但在当前 RISC-V 架构下代码是逐块解密串行的，存在性能瓶颈。

优化方案：

- 专用汇编实现：基于 RISC-V 的 Zvkned 指令集，反合 AES-128-CBC 加解密汇编优化，充分利用硬件指令加速计算。
- 并行解密处理：针对 CBC 模式解密可并行化特点，进一步优化解密流程，采用批处理策略，增加 6 块并行解密循环，多数据块并行解密提升处理能力。

（3）SM2 算法优化

核心瓶颈：SM2 算法的性能主要受限于大数模运算，C 语言通用实现缺少素数快速约简算法优化，以及 RISC-V 架构指令集加速，存在性能瓶颈。

优化方案：

- 算法优化：依据 SM2 的算法特点，使用素数快速约简算法优化通用 C 代码的性能瓶颈。
- 寄存器优化分配：合理运用 RISC-V 通用寄存器，减少内存访问，将运算数据尽可能保留在寄存器中，降低访问延迟。
- 指令流水线调度：结合 RISC-V 处理器流水线特征，通过指令重排与循环展开，提升指令级并行度与吞吐率。

应用场景

OpenSSL 密码学算法库的 RISC-V 架构性能优化主要应用于各类需要高效密码学计算的场景，为核心安全工作负载提供全面的算法加速支持。它使得基于 RISC-V 的设备能够在保持安全性的同时显著提升各类密码学运算效率，满足不同应用场景对算法性能的差异化需求，有效解决了 RISC-V 生态系统在商用密码学计算中的性能瓶颈问题。

（1）RSA 算法优化应用场景

关键应用领域：

1）数字签名与证书验证：电子商务平台、金融交易系统、软件分发系统中的数字证书链验证，优化后的 RSA 算法能够显著加快 HTTPS 握手过程中的证书验证速度，降低用户访问延迟。

2）安全密钥交换：VPN 网关、SSH 服务器、TLS 终端等安全通信设备中的 RSA 密钥交换过程，性能提升可减少连接建立时间，提高高并发场景下的连接处理能力。

3）身份认证系统：企业单点登录、身份管理平台中的 RSA 签名验证，优化后能够支持更高的认证请求并发量，提升系统整体响应性能。

（2）AES-128-CBC 算法优化应用场景

关键应用领域：

1）数据传输加密：网络通信设备、路由器、防火墙中的数据包加密传输，优化后的 AES-128-CBC 能够加速数据加密传输。

2）存储数据保护：数据库加密、文件系统加密、云存储服务等场景中的数据加密存储，性能提升可减少加密操作对业务系统的影响，提高数据存取效率。

3）批量数据处理：大数据平台、数据备份系统、日志加密等需要处理大量加密数据的场景，优化后能够显著提升批量数据的加解密处理速度，降低数据处理总耗时。

(3) SM2 算法优化应用场景

关键应用领域：

1）数字签名与证书验证：电子商务平台、金融交易系统、软件分发系统中的数字证书链验证，优化后的 SM2 算法能够显著加快国密 HTTPS/TLS 中的证书验证速度，降低用户访问延迟。

2）安全密钥交换：VPN 网关、SSH 服务器、TLS 终端等安全通信设备中的 SM2 密钥协商过程，性能提升可减少连接建立时间，提高高并发场景下的连接处理能力。

3）身份认证系统：企业单点登录、身份管理平台中的 SM2 签名验证，优化后能够支持更高的认证请求并发量，提升系统整体响应性能。

Golang Backport RVA23 Profile 支持

功能描述

为 openEuler 24.03 LTS SP3 中的 Golang 1.21 版本引入 GORISCV64 环境变量，以及包括 RVA23 在内的各个级别的 RVA Profile 支持，使得 openEuler 中的 Golang 可以支持更多高级 RISC-V Profile 中的指令拓展（Zbb、V），进一步优化 Golang 在 RISC-V 下的性能表现。

应用场景

启用 RVA23 Profile 后，使得 Golang 在支持的 RISC-V CPU 上使用更多高级的指令拓展，有效提升了 Golang 在 RISC-V CPU 下的性能表现。经过测试 Golang math 数学运算综合各项耗时 -21.75%，math/bits 数学位运算综合各项耗时 -32.65%，预期在实际各个应用场景当中也能活动性能提升收益。

openssl Backport 主线 RISC-V 架构的 SHA-2 汇编优化

功能描述

openEuler RISC-V 24.03 LTS SP3 中的 openssl 基线版本为 3.0.12 ，而当前 openssl 的最新版本已经到了 3.6 系列。将上游主线版本中已合并的部分密码学汇编优化算法 Backport 到 3.0.12 中，可以进一步优化 openEuler RISC-V 24.03 LTS SP3 下 openssl 在 RISC-V 下的性能表现。

在 Backport 主线 RISC-V 架构的 SHA-2 汇编优化后，openssl 在执行 SHA-2 算法时会先根据当前 CPU hwprobe 判断当前 CPU 支持的指令拓展情况，再根据当前 CPU 的选择对应的优化函数，从而在执行 SHA-2 算法时获得更好的性能表现。

应用场景

SHA-2 算法的核心应用场景如下：

1. 数据完整性验证

目的：保证数据未被篡改。

典型应用：计算文件、消息的哈希值。验证软件下载是否完整、数据传输是否准确。

2. 数字签名基础

目的：保证签名效率和安全性。

典型应用：对消息的哈希值进行签名，而非消息本身。作为 SM2/RSA 等签名算法的前置步骤。

3. 密码安全存储

目的：保证密码的机密性。

典型应用：系统数据库不存储明文密码，而是存储加盐后的密码哈希值。验证时对比哈希值即可。

openssl Backport 主线 RISC-V 架构的 MD5 汇编优化

功能描述

openEuler RISC-V 24.03 LTS SP3 中的 openssl 基线版本为 3.0.12 ，而当前 openssl 的最新版本已经到了 3.6 系列。将上游主线版本中已合并的部分密码学汇编优化算法 Backport 到 3.0.12 中，可以进一步优化 openEuler RISC-V 24.03 LTS SP3 下 openssl 在 RISC-V 下的性能表现。

在 Backport 主线 RISC-V 架构的 MD5 汇编优化后，openssl 在执行 MD5 算法时会先根据当前 CPU hwprobe 判断当前 CPU 支持的指令拓展情况，再根据当前 CPU 的选择对应的优化函数，从而在执行 MD5 算法时获得更好的性能表现。

应用场景

MD5 算法的核心应用场景如下：

1. 非密码学安全校验

目的：快速检查数据意外错误。

典型应用：用于网络传输、文件系统的非恶意错误检测。

2. 遗留系统兼容

目的：保证与旧系统、旧协议的兼容。

典型应用：在一些非安全的旧有文件校验、数据去重场景中可能遇到。

OpenJDK21 Backport 上游 RV 架构优化

Java 作为企业级应用和大规模分布式系统的核心编程语言，长期支撑着金融、电信、互联网等关键领域的后端基础设施。针对 OpenJDK 主干（master）中已实现但尚未包含于 OpenJDK 21 的 RISC-V 关键优化，我们通过代码反合，将部分 RVA23 Profile 相关的指令扩展成功集成至 OpenJDK 21；同时反合了大部分加解密与编码算法的汇编级优化，以及部分核心 Java 操作的 intrinsic 实现。上述工作显著提升了 OpenJDK 21 在 RISC-V 平台上的运行效率，覆盖网络、存储、密码学及并发等关键场景，为 openEuler 生态在 RISC-V 架构上提供高性能、生产级 Java 运行时能力奠定了坚实基础。

功能描述

RVA23 Profile 相关的指令扩展反合

将 OpenJDK 主干中对 RVA23 Profile 所定义的 Zfa、Zacas、Zabha、Zvkn、Zicond 等 RISC-V 指令扩展的支持反向合并至 OpenJDK 21。在此基础上，集成 Linux hwprobe 系统调用机制，在 JVM 启动及 JIT 编译阶段动态探测处理器实际支持的指令集能力，确保仅在硬件明确支持的前提下启用对应的优化代码路径；对于不支持相关扩展的 RISC-V 平台或探测失败的环境，系统自动回退至通用实现，完整保持功能正确性与跨平台兼容性。

解密算法汇编优化反合

为 SHA1/2、ChaCha20、Poly1305、CRC32、Adler32、Base64 等核心加解密与编码算法提供了 RISC-V 专属的高性能汇编实现；结合指令扩展能力（如 Zvkn 向量加密扩展），充分发挥 RISC-V 向量与标量指令协同优势，在典型负载下实现数倍性能提升，并通过运行时能力检测确保仅在具备相应硬件支持的环境中激活优化路径。

核心 Java 操作 intrinsic 反合

对 Object.hashCode()、Math.copySign()、BigInteger 大整数运算及 fastlock 轻量级锁等高频 Java 操作实施 intrinsic 化改造，利用 RISC-V 特定指令生成更紧凑高效的机器码；该优化在保持 Java 语义完全兼容的前提下，显著降低方法调用开销与计算延迟。

应用场景

高性能 Java 服务运行环境

在基于 openEuler 的 RISC-V 云服务器上，Java 应用广泛承担 Web 服务、微服务治理、中间件等关键角色。本次 OpenJDK 21 优化一定程度提升其运行效率，优化了以下场景：

- 高并发 Web 应用：fastlock 优化及原子操作扩展（Zacas、Zabha）降低锁竞争开销，hashCode 和 BigInteger 加速提升对象处理与身份校验性能，显著改善 API 响应延迟；
- 微服务通信与序列化：Base64、CRC32、Adler32 等编码/校验算法的汇编优化，加速 JSON/XML 序列化、gRPC/HTTP 通信及服务间数据校验。

安全与网络密集型系统

随着 RISC-V 在边缘安全网关、可信执行环境等场景的落地，加密与认证计算成为性能瓶颈。本优化在以下场景均有涉及：

- TLS/SSL 加速：SHA1/2、ChaCha20、Poly1305 等算法的向量化汇编实现，显著提升 HTTPS、QUIC 等安全协议的握手与数据加密吞吐；
- 文件系统与存储校验：CRC32 与 Adler32 优化加速对象存储（如 MinIO）的块校验、日志写入及快照生成。

技术规格

（1）支持版本

- 架构支持：RISC-V 64 位架构（需支持 Zfa、Zacas、Zabha、Zvkn、Zicond 扩展）；
- 操作系统：Linux 6.6 及以上内核版本；
- OpenJDK 版本：21.0.9。

（2）安全与合规

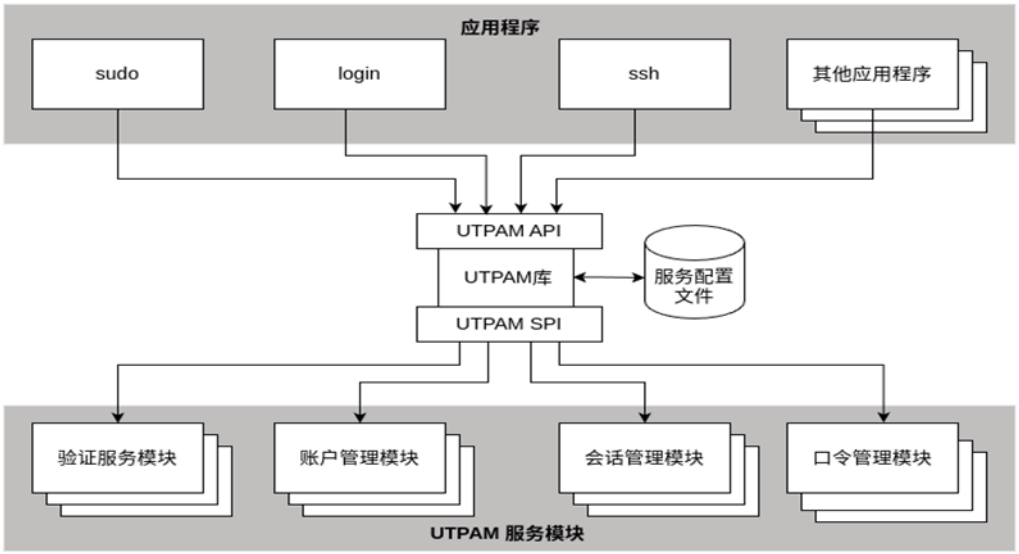
反合内容严格遵循 RVA23U64 架构规范要求，确保了硬件兼容性和安全性。相关扩展与优化代码在激活前进行严格的硬件特性检测，避免在不支持的硬件上引发未定义行为或性能回退。

utpam：基于 Rust 开发的身份认证模块

utpam 是一种用于 Linux 系统的认证框架。它允许系统管理员定义系统中的不同服务的认证机制，并且可以根据需要组合多种认证方式。utpam 通过提供一个统一的接口来处理认证相关的工作，简化了应用程序对用户身份验证的过程。

功能描述

下图为 utpam 工具的整体系统结构，libutpam 是 utpam 的核心库，它根据应用程序提供的配置文件进行认证的初始化，配置文件默认位于/etc/utpam 目录下，文件以服务名称命名。utpam 向上为应用程序提供 API 接口选择认证类型，认证类型共有四类，分别为用户验证、账户管理、会话管理和口令管理；utpam 向下为开发者提供与 API 对应的 SPI 接口来实现具体的认证过程。



应用场景

UTPAM 是针对 Linux PAM 的 Rust 扩展套件，提供可插拔的认证模块与工具库，可在登录、鉴权与会话管理等阶段按需加载。典型场景包括：

安全加固：在特定阶段默认拒绝或严格校验，阻断不合规的认证路径。

特权管控：对 root/高权限账户执行快速放行或额外检查，降低运维摩擦。

审计与提示：在认证流程输出上下文信息（用户、TTY、服务、来源），提升可观测性。

异构集成：通过稳定的 C 接口与头文件接入传统 PAM/C 系统，同时复用 Rust 公共库。

适用人群与需求：需要分阶段控制认证行为、增强审计提示、对特权账户施策，以及在现有 PAM/C 环境引入更安全可靠的 Rust 实现的团队与系统。

secScanner：操作系统安全加固组件

secScanner 是一款 Linux 操作系统安全扫描工具，旨在为操作系统提供安全管理、漏洞扫描、rootkit 入侵检测等功能。

功能描述

secScanner 基于“3 核心能力+3 公共能力”的技术架构，建设全面有效、实时自动、灵活易用的主动防护能力。

- **安全管理**：操作系统已有安全能力的使能度管理，针对操作系统安全配置复杂、运维人员安全意识不足、不同场景下安全配置需求不同等问题，安全管理功能可支持多基线选择，亦可根据需求定制安全基线，通过配置文件中预设的参数细粒度控制安全配置项，一键进行安全检测、安全加固、配置还原的功能。

- **漏洞扫描**：全面和及时应对操作系统组件已知 CVE 威胁，针对操作系统组件存在已知的 CVE 漏洞，潜在攻击者利用已知的攻击路径发起攻击。漏洞扫描功能支持从 openEuler 安全中心下载安全公告、CVE 公告保存在本地漏洞数据库，对系统组件进行全量扫描或者重点组件轻量化定向扫描，报告系统当前存在风险的组件，并给出升级建议。
- **入侵检测**：泛化性应对操作系统未知威胁，针对攻击者通过各种未知手段对系统进行入侵，入侵检测功能调用了 secDetector 工具对系统潜在的恶意 rootkit 模块进行检测，并给出清理建议。

应用场景

secScanner 安全加固工具主要应用于服务器安全运维场景。能够提供自动化的安全运维，一键检测当前系统中的薄弱配置项，并提供按照预置基线进行配置项加固的能力。有效防止运维人员安全意识不足产生的安全风险，减少运维人力，保护系统安全。

VMAnalyzer：轻量级虚拟化性能监控组件

VMAnalyzer 是一款轻量级的虚拟化监测分析工具，围绕如下两个方面进行设计：

虚拟化环境监测：通过实时监测虚拟机的 CPU、内存、磁盘 I/O 等关键性能指标，发现性能瓶颈。

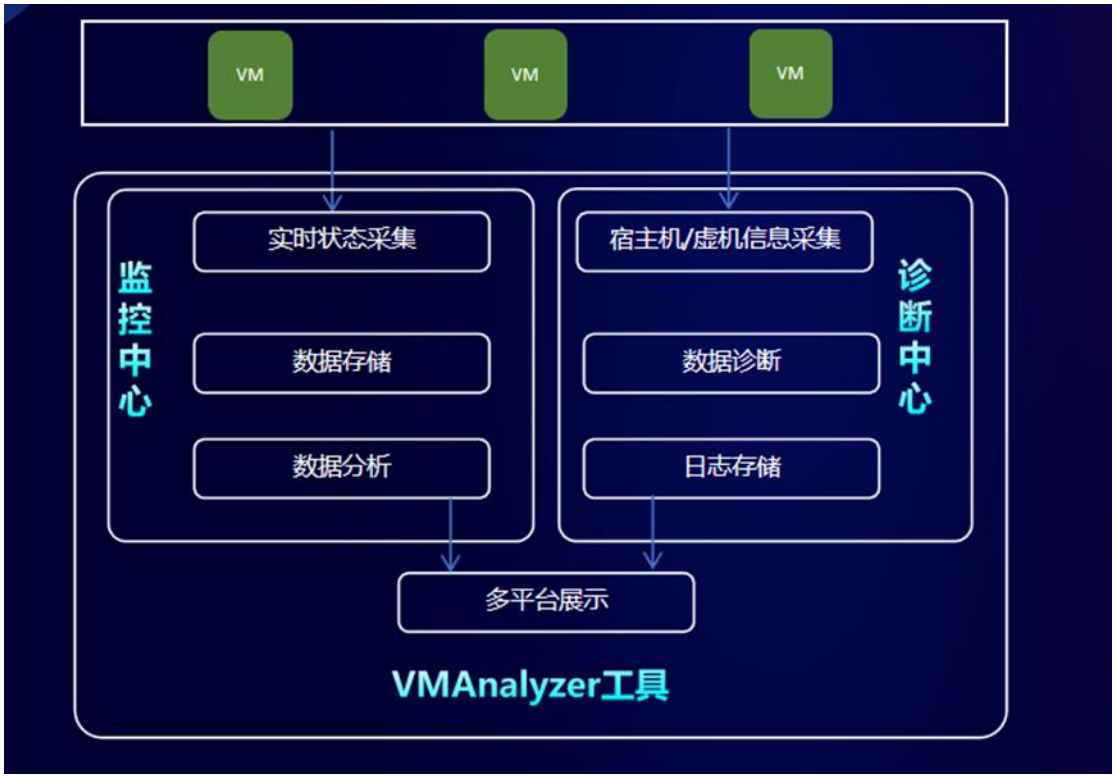
高可靠性保障：通过分析虚拟机的 qemu 进程、物理环境配置等信息提供高可靠性的虚拟机维护方案，预测并发现潜在的故障风险。

功能描述

VMAnalyzer 包括监控中心、诊断中心两部分，架构如下：

监控中心：实时采集虚机的数据，能够细粒度的分析虚拟机的运行状况，检测结果的可灵活通过 console、OPS、DW 等多平台展示各个云主机数据。

诊断中心：针对虚拟化层面的问题，通过执行命令对虚拟化层进行深度诊断，通过配置分析、状态分析，虚机进程消耗分析等，帮助用户分析虚拟化层面的各类问题，最终通过日志文件展示出来。



应用场景

VMAnalyzer 是一款轻量级的虚拟化性能监测分析工具，主要应用于云计算环境，能够细粒度的分析虚拟机的运行状况和性能，可轻松识别性能问题和瓶颈，可以帮助用户维护具有高性能和高可靠性的虚拟机。

9. 著作权说明

openEuler 白皮书所载的所有材料或内容受版权法的保护，所有版权由 openEuler 社区拥有，但注明引用其他方的内容除外。未经 openEuler 社区或其他方事先书面许可，任何人不得将 openEuler 白皮书上的任何内容以任何方式进行复制、经销、翻印、传播、以超级链路连接或传送、以镜像法载入其他服务器上、存储于信息检索系统或者其他任何商业目的的使用，但对于非商业目的的、用户使用的下载或打印（条件是不得修改，且须保留该材料中的版权说明或其他所有权的说明）除外。

10. 商标

openEuler 白皮书上使用和显示的所有商标、标志皆属 openEuler 社区所有，但注明属于其他方拥有的商标、标志、商号除外。未经 openEuler 社区或其他方书面许可，openEuler

11. 附录

附录 1：搭建开发环境

环境准备	地址
下载安装 openEuler	https://openeuler.org/zh/download/
开发环境准备	https://atomgit.com/openeuler/community/blob/master/zh/contributors/prepare-environment.md
构建软件包	https://atomgit.com/openeuler/community/blob/master/zh/contributors/package-install.md

附录 2：安全处理流程和安全批露信息

社区安全问题披露	地址
安全处理流程	https://atomgit.com/openeuler/security-committee/blob/master/docs/zh/vulnerability-management-process/security-process.md
安全披露信息	https://atomgit.com/openeuler/security-committee/blob/master/docs/zh/vulnerability-management-process/security-disclosure.md
安全保障策略总纲	https://atomgit.com/openeuler/security-committee/blob/master/security-strategy-overview.md