OpenEuler x Compiler SIG

Compiler SIG 24年工作进展及下一步规划介绍及讨论

华为

谢志恒

目录

毕昇JDK在java启动、JIT增强、堆数据安全等构建新能力

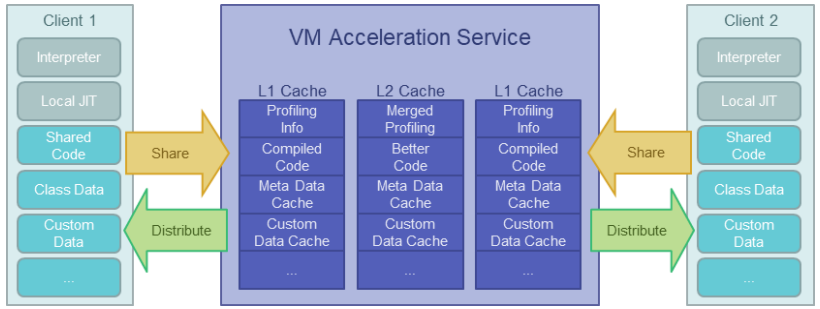
OpenEuler Compiler SIG

问题场景：

随着云计算的发展，越来越多的应用被部署在云场景下，传统VM应用面临着较为严重的启动性能问题。为此毕昇设计一个能够兼顾加速比、通用性、易用性的启动加速方案。

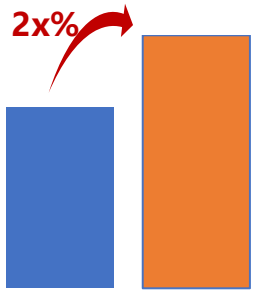
关键技术：Jbooter分布式JIT

- 提供了一个分布式VM加速服务，支持多级缓存
- 缓存、汇聚、共享客户端VM运行时数据
- 缓存数据包括代码、类数据、Profiling等



JBoosterVM加速服务：缓存、共享、迭代多种缓存

预期效果：



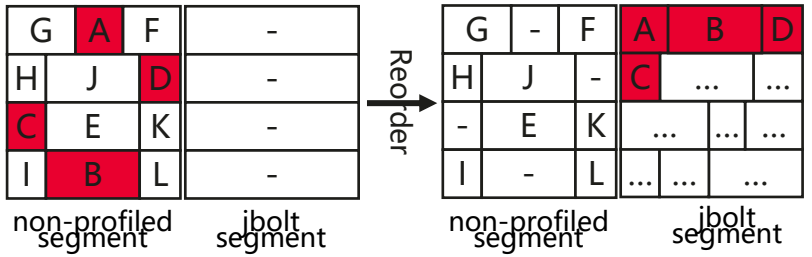
应用启动性能

问题场景：

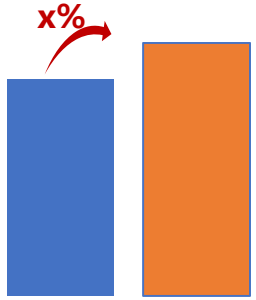
长周期运行的Java服务型应用的场景中，存在较多的icache load miss和iTLB load miss问题。针对此类问题毕昇JDK对iCache和iTLB的利用率进行优化，提升应用峰值性能。

关键技术：Jbolt 开发中

- 运行时采样：在虚拟机运行时使用JFR采样并追踪热点编译方法的调用链，在线生成函数热力排序
- JIT Cache布局优化：基于获取的热点调用链call graph，基于C3算法得到方法间较优布局，时空分布->热点分布



预期效果：



应用峰值性能

问题场景：

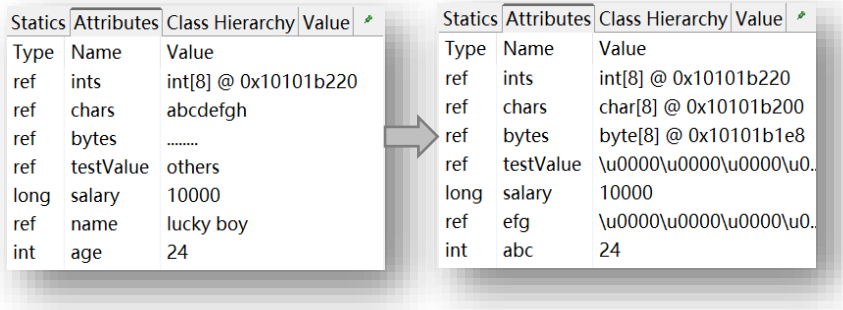
云应用（支付、云空间、运动健康、手表）用户隐私数据（如账号/密码、医疗信息等敏感数据）保护在云侧。在需要借助堆内存文件定位问题时，这些信息可能会被暴露。为此毕昇JDK设计一套高效、便捷的匿名化方案。

关键技术：HeapDump匿名化

- 匿名化参数灵活使用：支持应用启动时添加JVM参数，也支持运行时使用jmap、jcmd导出匿名化堆
- 支持多种匿名化模式：支持basic、name、full、annotation、diyrules等多种模式。

对象	int[]	char[]	byte[]	Symbol
模式	off	basic	annotation	
	full	nams	diryrules	
工具	jcmd	jmap	VM参数	

预期效果：



Statics	Attributes	Class Hierarchy	Value
Type	Name	Value	
ref	ints	int[8] @ 0x10101b220	
ref	chars	abcdefgh	
ref	bytes	.....	
ref	testValue	others	
long	salary	10000	
ref	name	lucky boy	
int	age	24	

Statics	Attributes	Class Hierarchy	Value
Type	Name	Value	
ref	ints	int[8] @ 0x10101b220	
ref	chars	char[8] @ 0x10101b200	
ref	bytes	byte[8] @ 0x10101b1e8	
ref	testValue	\u0000\u0000\u0000\u0...	
long	salary	10000	
ref	efg	\u0000\u0000\u0000\u0...	
int	abc	24	

毕昇JDK发挥软硬协同，构建openEuler + JDK差异化能力

OpenEuler Compiler SIG

问题场景：

- **数据压缩：**数据压缩技术能够有效降低数据存储及传输成本，在Java领域里有广泛的应用。尤其在一些数据存储、http等场景压缩消耗的CPU占比甚至高达50%+。毕昇JDK充分发挥鲲鹏亲和性优势，使能KAE硬件加速，大幅度提升压缩解压缩性能。
- **加解密：**加解密算法在数据加密传输、数据库加密存储、安全证书、数字签名等场景中使用，在Java领域中尤其HTTPS加解密算法耗时占比很大。毕昇JDK充分发挥鲲鹏亲和性优势，使能KAE硬件加速，大幅度提升加解密缩性能，进而缩短HTTPS传输时间。

关键技术：

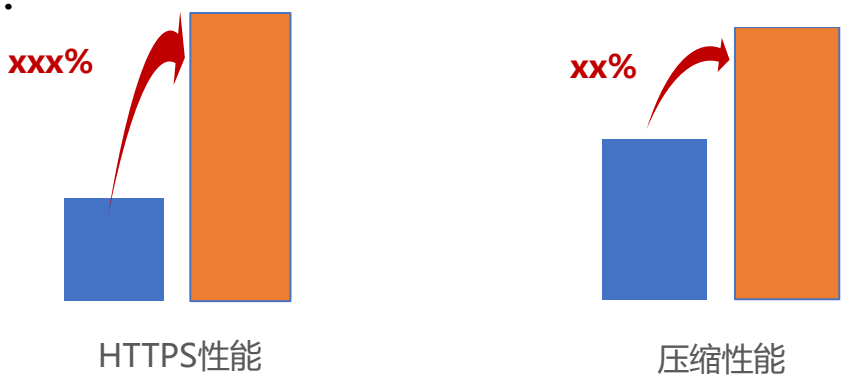
数据压缩： KAE zip

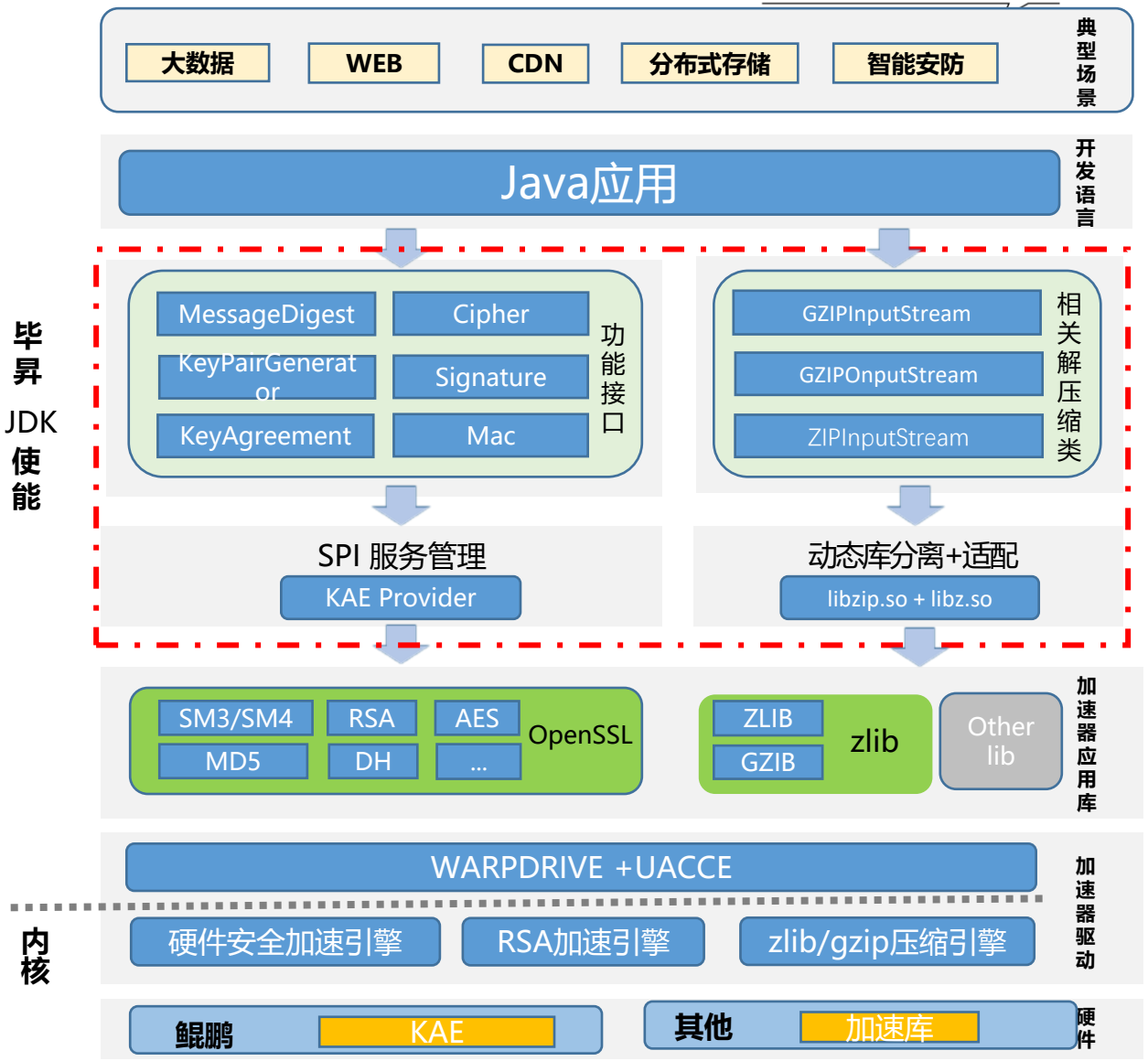
- 通过动态库分离及API适配完成JVM对kae zip库的依赖
- KAE 压缩引擎当前支持 ZLIB、GZIP算法，压缩比约为2，压缩速度高达7GB/s

加解密： KAE 加解密

- 通过JCE扩展机制将KAE加解密算法集成到JDK KAEProvider
- 支持摘要、加解密、RSA签名、EC/ECDH秘钥生成等，算法包括RSA/SM3/SM4/DH/MD5/AES 等多种

预期效果：





毕昇JDK增强运维工具功能，提升java业务场景运维效率

OpenEuler Compiler SIG

问题场景：

应用发生Crash时，缺乏部分信息难以定位问题原因。毕昇JDK8增强hs_err日志，在日志中提供更多的信息，如系统资源限制、堆栈源信息等以供定位问题

关键技术： hs_error log增强

- 打印更多fd、线程、内存等系统资源
- 打印SafepointTimeout 信息
- 打印堆栈信息中的源文件名和行号

thread-max	max_map_count	pid_max	
Virtual Size	RSS	Swapped	C-Heap
源文件名及行号	SafepointTimeout		

预期效果：

```
/proc/sys/kernel/threads-max (system-wide limi
2056454


/proc/sys/vm/max_map_count (maximum number of
65530


/proc/sys/kernel/pid_max (system-wide limit on
131072
```

```
Process Memory:
Virtual Size: 41849012K (peak: 41849012K)
Resident Set Size: 10510152K (peak: 10510184K)
Swapped out: 0K
C-Heap outstanding allocations: 145836K, retair
```

问题场景：

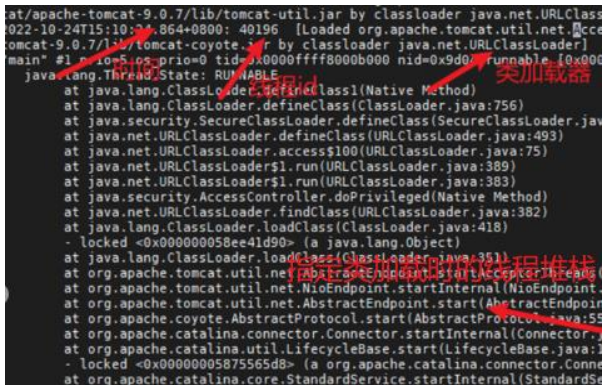
在定位类加载相关问题时，目前TraceClassloading输出信息较少，只能显示是否被加载、从哪个文件被加载。毕昇JDK8提供更多信息辅助定位类加载问题

关键技术： TraceClassLoading增强 | jcmd增强0

- Trace打印更多的类加载信息
- 增强jcmd功能打印类信息、类加载器信息
- …

类加载时间	类加载线程 id
类加载器	类加载线程堆栈
VM.classes	VM.classloaders

预期效果：



问题场景：

Linux环境常用perf工具抓取热点、火焰图等信息用以性能调优，但是perf抓取Java应用很多函数名无法解析，不利于分析。毕昇JDK8提供一个能够方便快捷、易用性高的方案更好的利用perf

关键技术： jcmd增强1

- 提供了一个jcmd命令jcmd <pid> Compiler.perfmap
- 记录虚拟机函数汇编信息到/tmp/perf-$PID.map文件
- perf解析/tmp/perf-$PID.map文件输出更加直观的Java方法信息
- …

预期效果：

```
libjvm.so          [.] jni_CallStaticVoidMethod
libjvm.so          [.] jni_invoke_static
libjvm.so          [.] JavaCalls::call_helper
perf-117434.map    [.] NMTMain.main([Ljava/lang/String;)V
libc-2.28.so       [.] thread_start
libpthread-2.28.so [.] start_thread
libjvm.so          [.] java_start
libjvm.so          [.] JavaThread::run
libjvm.so          [.] JavaThread::thread_main_inner
libjvm.so          [.] thread_entry
libjvm.so          [.] JavaCalls::call_virtual
libjvm.so          [.] JavaCalls::call_virtual
libjvm.so          [.] JavaCalls::call_helper
```

问题场景：

Java应用存在部分不易观测的内存区域，如Metaspace内存碎片、Native-Heap内存碎片，且其中Native-Heap部分缺乏有效控制手段。毕昇JDK8提供一个能够观测相关区域的手段，且可以直接通过Java端控制Native-Heap内存

关键技术： jcmd增强2

- 提供了一个jcmd命令jcmd <pid> VM.metaspace，且在Metaspace OOM时自动打印元数据区的内存布局
- 提供了一个jcmd命令jcmd <pid> System.native_heap_info，可以输出进程Native-Heap使用详情
- 提供了一个jcmd命令jcmd <pid> System.trim_native_heap，可以自行修剪Native-Heap内存

预期效果：

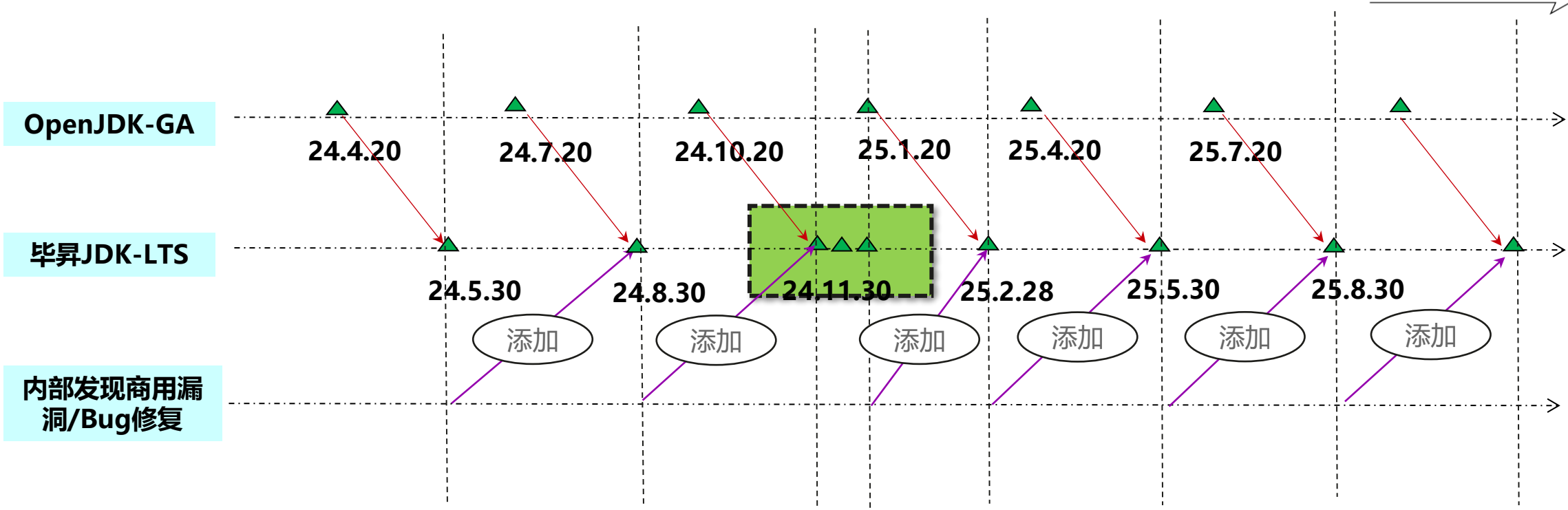
```
Summary:
 Total class loaders=    75 capacity=   4635.00KB used=   3823.48KB free=    811.52KB waste=      0.02KB
                         Metadata capacity=   4176.00KB used=   3408.42KB free=    767.58KB waste=      0.02KB
                         Class data capacity=    459.00KB used=    415.06KB free=     43.94KB waste=      0.00KB

For anonymous classes=   73 capacity=    149.00KB used=     93.60KB free=     55.40KB waste=      0.02KB
                         Metadata capacity=     76.00KB used=     53.09KB free=     22.91KB waste=      0.02KB
                         Class data capacity=   73.00KB used=     40.52KB free=     32.48KB waste=      0.00KB
```

```
</heap>
<total type="fast" count="0" size="0"/>
<total type="rest" count="164" size="17605026"/>
<total type="mmap" count="136" size="140361728"/>
<system type="current" size="26988544"/>
<system type="max" size="26988544"/>
<aspace type="total" size="26988544"/>
<aspace type="mprotect" size="26988544"/>
</malloc>
```

```
jcmd 39138 System.trim_native_heap
39138:
Trim native heap: RSS+Swap: 10148M->10044M (-106388K)
```

毕昇JDK版本发布周期及CVE漏洞更新机制

OpenEuler Compiler SIG



例行发布时间点：

◆ OpenJDK小版本升级GA的时间基本固定在每年的1/4/7/10月左右，毕昇JDK一般会在随后的1个月内发布对应的GA版本

漏洞/Bug修复策略：

◆ 毕昇JDK会在OpenJDK 漏洞修复的基础上，加上商用自发现的漏洞、Bug修复；

◆ 如遇严重问题，会紧急发布多个补丁版本

严重bug紧急发布多个修复版本（此处举例）

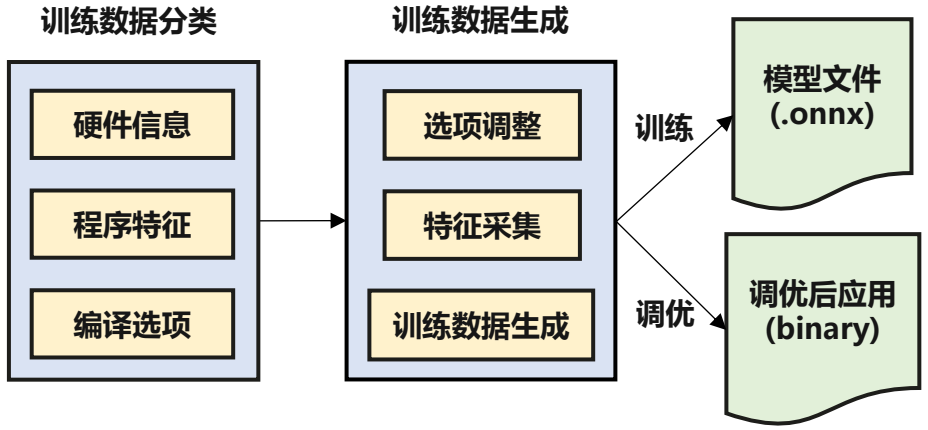
添加 新发现的漏洞（但社区未解决的）修复添加

目录

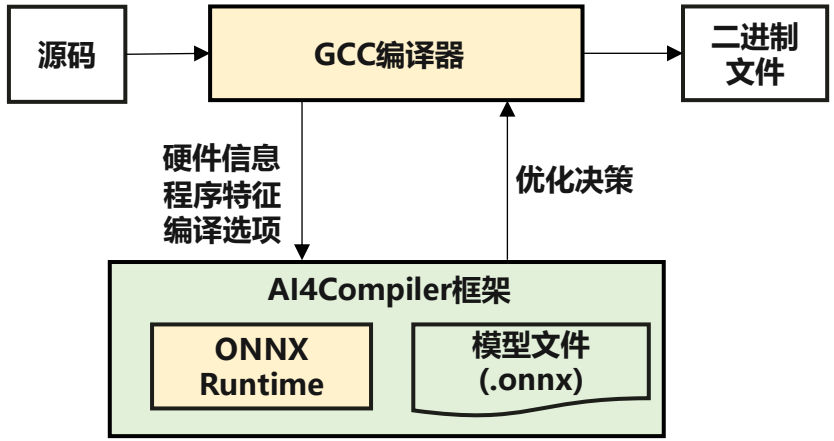
AI for Compiler关键技术与进展

OpenEuler Compiler SIG

Autotuner调优工具（模型训练）



AI辅助编译优化（模型推理）



业务挑战

软硬件环境发生变更时，旧场景开发的启发式编译优化算法普适性较差

- 当硬件架构或者软件业务场景发生变更时，由于需要人工根据新的负载条件，针对optimization pass的成本模型进行重新调整，导致调优时间较长。
- 传统optimization pass的开发周期较长，同时新的编译优化算法与编译器内已有的编译优化手段可能会产生冲突，无法满足业务目标。

AI for Compiler关键技术

白盒特征提取	基于应用IR、程序CFG图、硬件信息等内容，利用GNN或Transformer训练有效反映软硬件信息的特征，作为目标任务的输入。
ML成本模型	结合白盒特征，辅助典型的编译优化，例如，数据预取、基本块重排等，以实验数据驱动的ML模型替换编译器内的启发式成本模型，提升应用场景覆盖率及优化效果。
AI4C推理引擎	编译器在AI使能的优化遍内，通过AI4C框架推理训练好的AI模型，并把推理结果返回给编译器进行解析和后续优化。

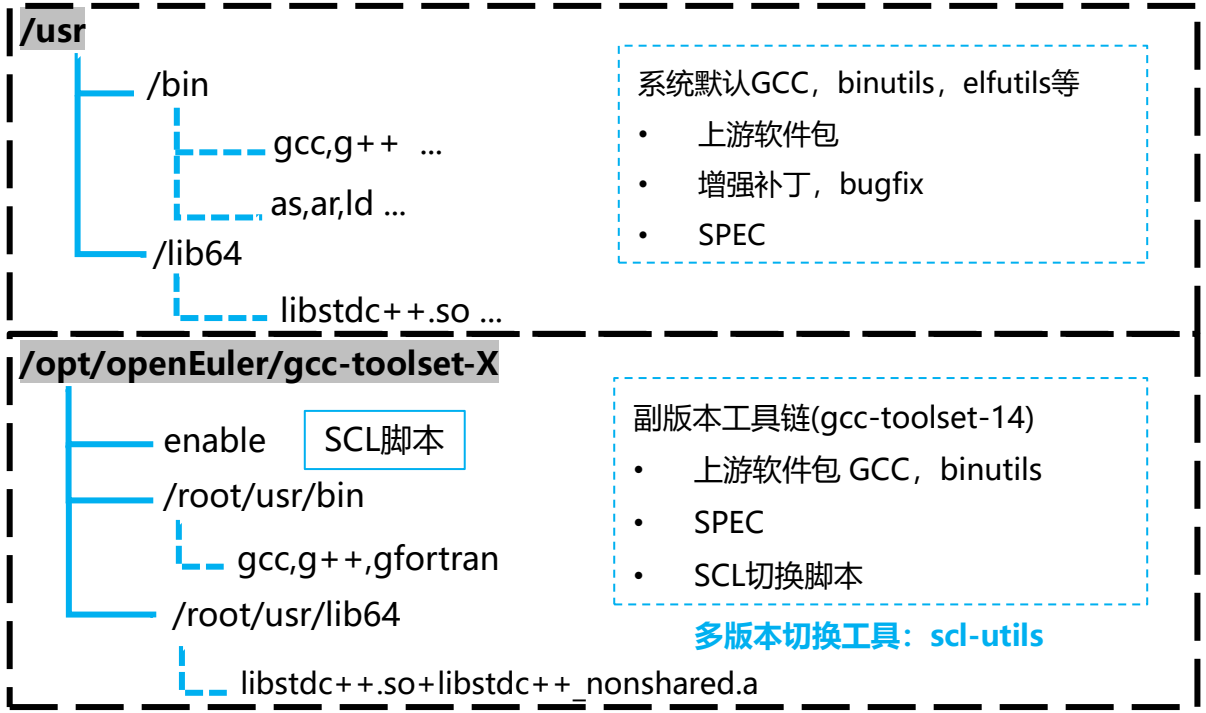
AI for Compiler当前进展

AI成本模型替换启发式成本模型，结合编译选项调优技术，数据库/大数据/分布式存储性能提升**5%~10%**

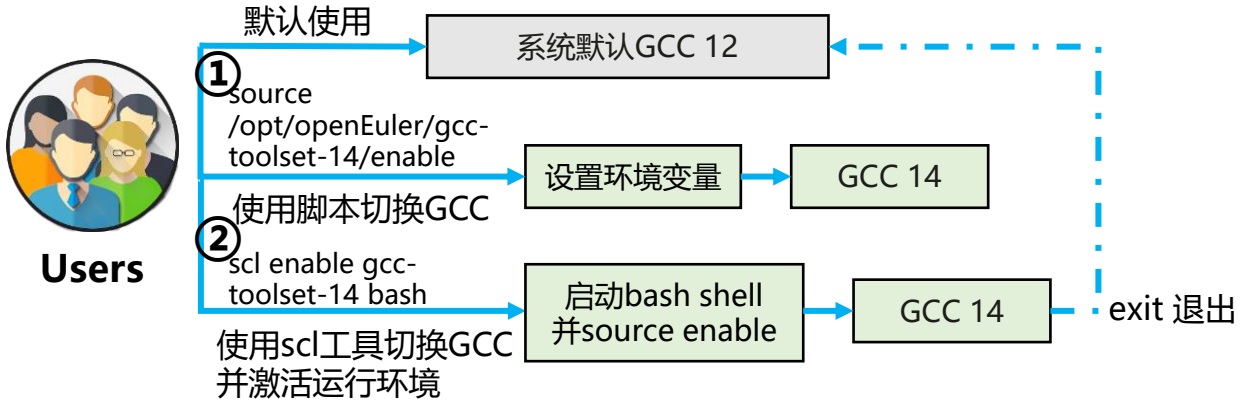
编译工具链多版本支持：devtoolset-14

OpenEuler Compiler SIG

- **引入GCC副版本编译工具链：devtoolset-14**



- **用户使用：提供两种切换方法**



版本支持进展与规划

- **openEuler 24.09版本：**
  - 已经发布devtoolset-14副版本编译工具链，包括多版本GCC 14.2.0 和Binutils-2.42 ，用户根据自己使用场景可以使用SCL进行切换：
  - 默认GCC路径: /usr/bin/gcc
  - 副版本GCC路径：/opt/openEuler/gcc-toolset-14/root/usr/bin/gcc
- **openEuler 24.X 版本：**
  - devtoolset-14副版本编译工具链中的GCC 14.2.0 会根据上游社区版本发布节奏，同步升级到GCC 14.3.0的稳定版本；
  - 根据用户需求进一步扩展后续GCC高版本的演进；

兼容性说明

- 主版本场景：正常使用OS中的默认版本GCC，运行默认GCC编译出的应用；
- 多版本场景：需要GCC高版本特性构建相关应用，使用scl工具将bash环境切换为gcc-toolset-X的高版本编译工具链的编译环境；
- **约束：**由于libstdc++的开发者非常重视libstdc++ ABI的兼容性，在libstdc++.so.6.0.0后，都是后向兼容的；因此，低版本GCC编译的程序可以运行在高版本GCC环境中；拆分动态库后，高版本GCC编译的程序也可以运行在低版本GCC的环境中。

LTO openEuler by default

OpenEuler Compiler SIG

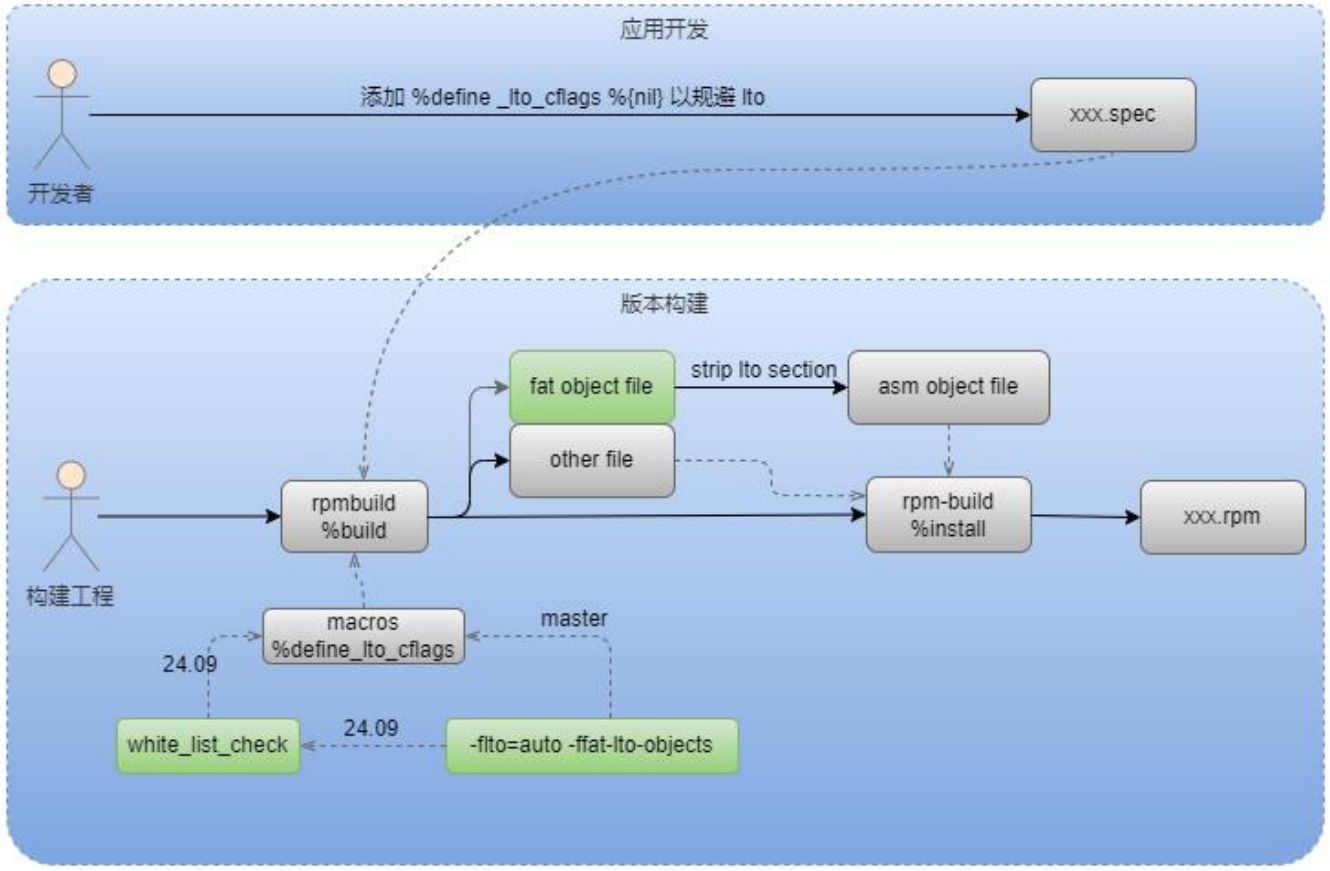
版本支持进展与规划

openEuler 24.09版本：

- 2409分支构建已通过openEuler-rpm-config结合白名单机制为 523 个应用使能 LTO；
- 减小生成的ELF可执行文件与共享库体积约300MB，占这 523 个应用的ELF可执行文件与共享库体积的14%；

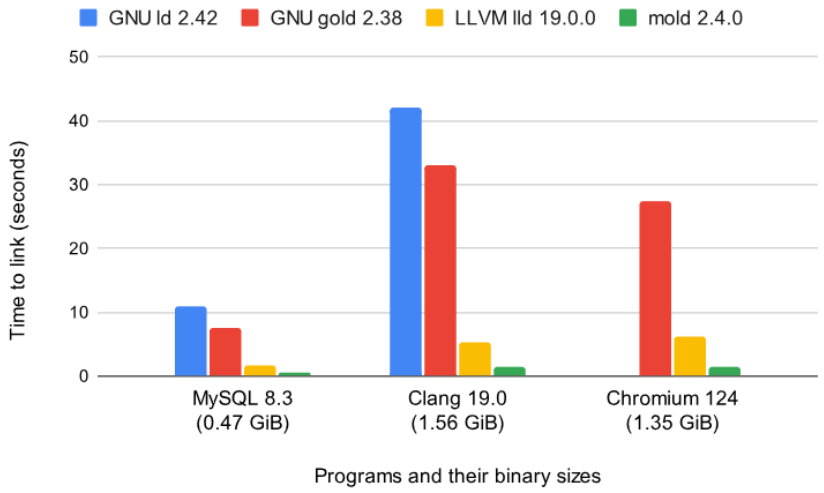
master分支：

- 作为未来长期演进的方向，为下个LTS大版本默认使能LTO做准备，当前master分支构建已通过openEuler-rpm-config**默认使能LTO（没有使用白名单机制）**；
- 若部分应用无法使能 LTO或不想使用LTO，开发者可以通过在 spec 中添加 %define _lto_cflags %{nil} 规避；
- 已通过**邮件、社区例会**等多种渠道同步LTO策略。



用户体验提升：降低构建链接时间，提升版本构建效率

OpenEuler Compiler SIG



Program (linker output size)	GNU ld	GNU gold	LLVM lld	mold
MySQL 8.3 (0.47 GiB)	10.84s	7.47s	1.64s	0.46s
Clang 19 (1.56 GiB)	42.07s	33.13s	5.20s	1.35s
Chromium 124 (1.35 GiB)	N/A	27.40s	6.10s	1.52s

不同链接器的链接速度差异（mold官方数据）

用户痛点：无论是工程构建还是开发者调试，典型应用的构建时长降低能够有效提升工程效率和用户开发体验；

方案设计

1、切换新型并行链接器：

- gold：切换GNU gold链接器引入未知风险比较小，也兼容GCC LTO，且当前该链接器已经比较成熟，对于链接时间长的应用提升效果明显，未来开发方向主要是使能重点应用后进行兼容性分析，链接器修改后的功能验证是重点方向；
- lld：llvm链接器，不兼容GCC LTO，当前社区中也没有lld适配GCC LTO的计划和方案，移植风险较高；
- mold：新型链接器，兼容GCC LTO，并且链接提升速度远超gold和lld，gcc12后也支持该链接器；MIT许可证限制弱于GPL和apache，商业使用不受限；风险点在于当前mold链接不支持内核和嵌入式场景，全量使能有风险；

2、融合多种链接方案：

- mold作为新型链接器，能大幅提升链接速度，对于编译过程中链接时间过长的应用提升明显；gold作为GNU默认支持的链接器，其兼容性已被验证；采用mold/gold混合使能工程方式，对于支持mold或使用mold提升明显的应用采用mold链接，其他应用使能gold链接的方案，降低openEuler工程10%整体构建时间。

3、反馈构建GCC：

- 根据GCC官网提供的数据，在X86中编译速度提升约为5%-7%，arm本地未复现出该效果；自举GCC需要修改默认GCC配置，风险在于需要评估整体收益和风险，作为备选方案；

当前进展：已初步在版本中使能并行链接器mold，正在联合工程团队验证构建效率提升情况

目录

LLVM LTS版本：打造优体验、高性能、易创新的LTS版本

OpenEuler Compiler SIG

互联网 | 运营商 | 金融 | 政府 | 安平 | …

编译器

GCC for openEuler | 毕昇JDK | 毕昇编译器 | LLVM LTS 版本

鲲鹏能力使能

鲲鹏硬件特性快速使能，软硬协同实现**基础性能/主力场景性能优化**，满足用户对编译器**能力需求**

openEuler编译器新选择

为oE社区开发者提供更多选择，可支持**更多语言/特性**，让oE拥抱更多异构算力，打造**异构编译标准**

依托openEuler社区，吸引行业、生态合作伙伴加入，构建LTS版本，打造国内编译器基线

优体验

Compiler CI，openEuler质量加固

发布稳定、易用的LTS版本

高性能

性能特性集成，多样算力支持、发挥

相比上游社区性能提升

易创新

架构解耦，易扩展

AI for Compiler、安全编译等新技术创新

LLVM LTS版本：打造优体验、高性能、易创新的LTS版本

OpenEuler Compiler SIG

优体验

Compiler CI

PR → 社区能力：默认触发门禁；评论触发门禁

LTS版本 → 内部能力：商用测试套；codedb；性能监控；…

LLVM多版本

openEuler 24.03 LTS SPX

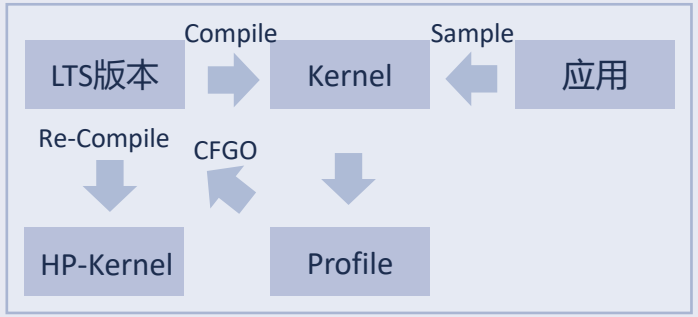
主版本：LLVM 17

多版本：LLVM 18；…

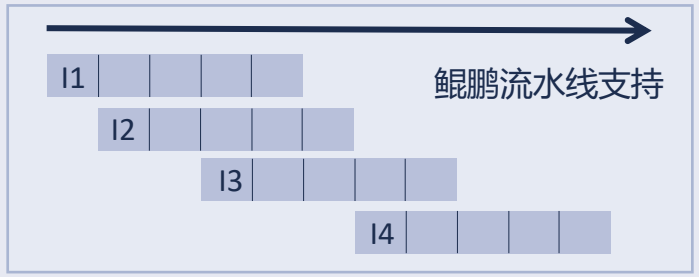
高风险bug监控回合　二进制包发布　…

高性能

Optimize for Kernel



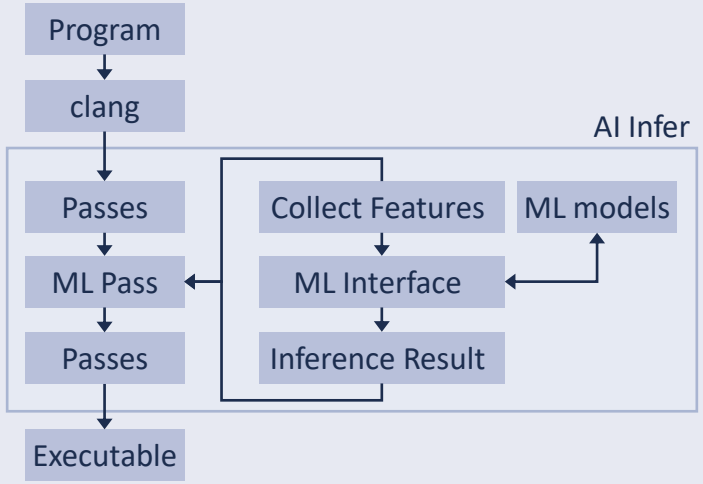
鲲鹏能力使能



鲲鹏持续使能　BOLT优化　…

易创新

ACPO



安全编译

-fstack-clash-protection　侧信道防护

弱内存序检测　栈保护　…

autotuner　AI models　…

OpenEuler x Compiler SIG

Thank You.

Compiler SIG 专注于编译器领域技术交流探讨和分享，包括 GCC/LLVM/OpenJDK 以及其他的程序优化技术，聚集编译技术领域的学者、专家、学术等同行，共同推进编译相关技术的发展。

毕昇编译公众号

Compiler 交流群小助手