鲲鹏软件迁移

目录

前言

本章节将从程序运行原理开始介绍，以了解软件迁移的背景与必要性。并通过软件迁移过程介绍迁移的五个步骤以及每个步骤的具体事项。

目标

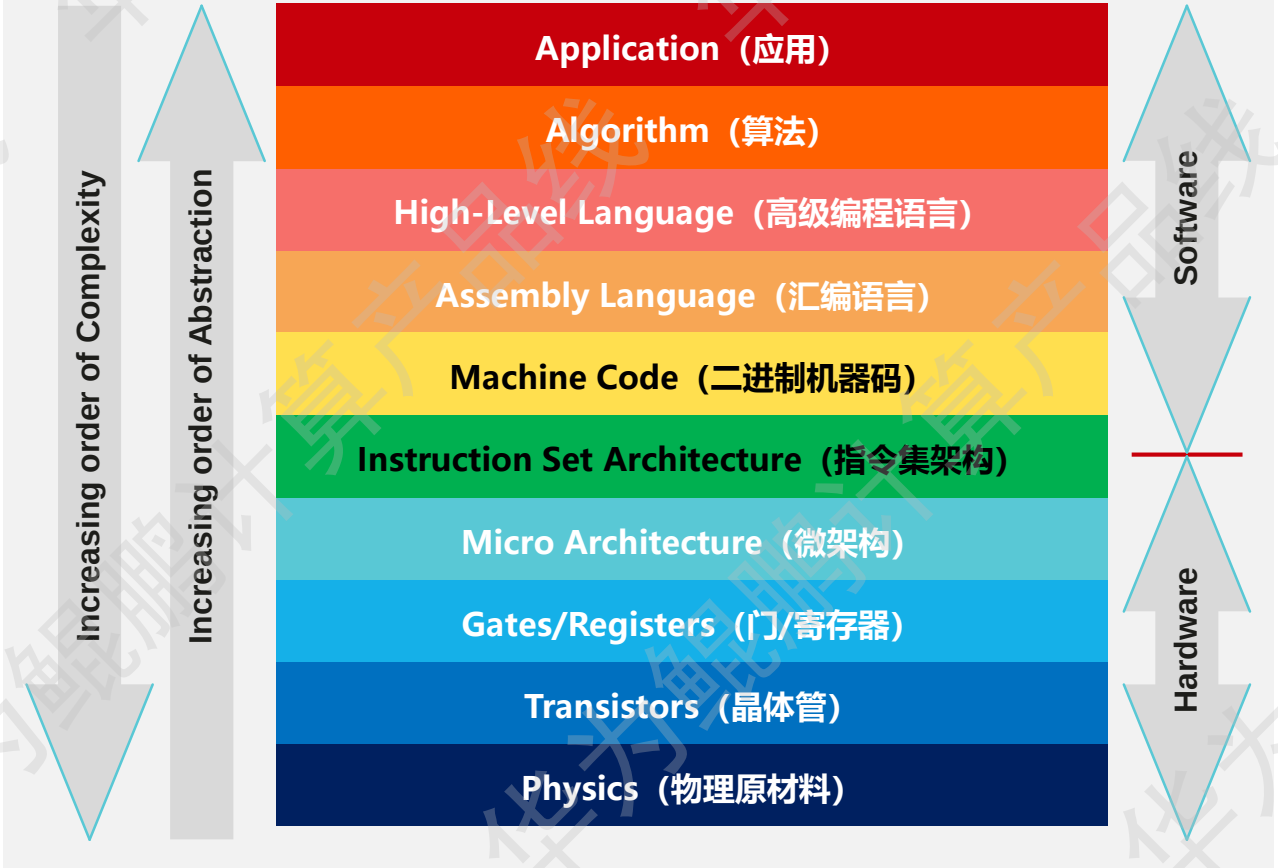
学完本章节后，您将能够：

- 了解程序运行原理
- 了解软件迁移至鲲鹏计算平台的过程

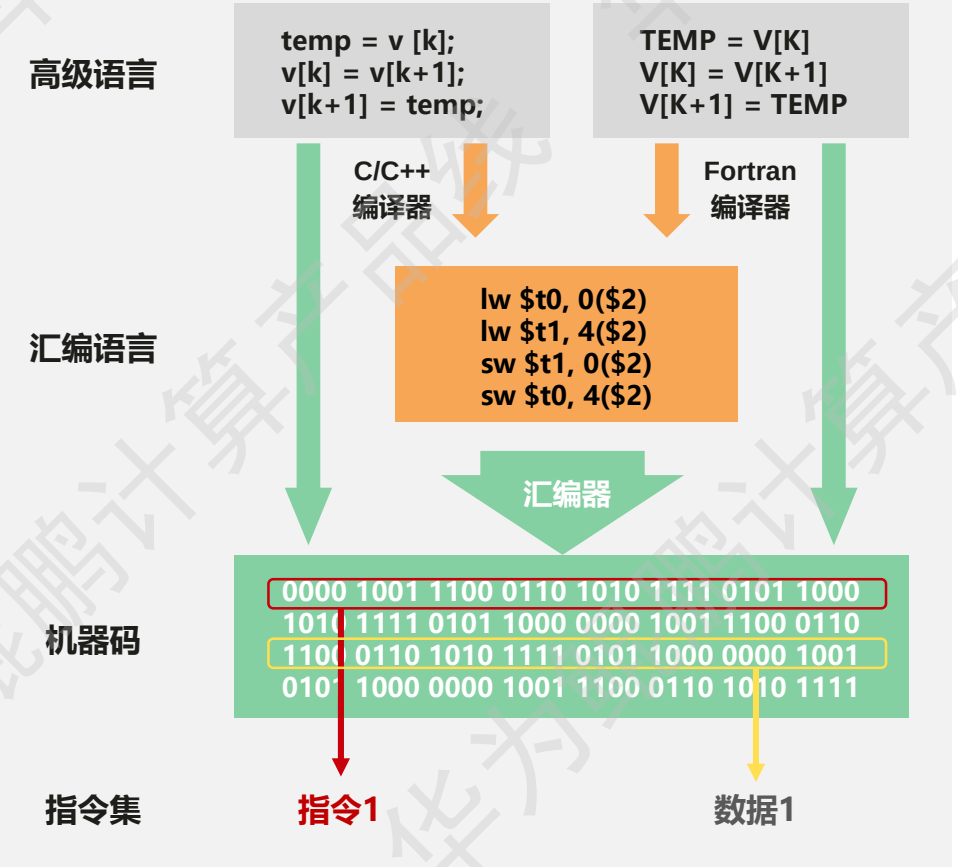
目录

计算技术栈与程序执行过程

计算技术栈



程序执行过程



鲲鹏处理器与x86处理器的指令差异

程序代码（C/C++）：

```
int main()
{
    int a = 1;
    int b = 2;
    int c = 0;
    c = a + b;
    return c;
}
```

编译

在x86编译后生成指令

汇编代码	指令	说明
mov -0x4(%rbp),%edx	8b 55 fc	从内存将变量a的值放入寄存器edx
mov -0x8(%rbp),%eax	8b 45 f8	从内存将变量b的值放入寄存器eax
add %edx,%eax	01 d0	将edx(a)中的值加上eax(b)的值放入eax寄存器
mov %eax,-0xc(%rbp)	89 45 f4	将eax寄存器的值存入内存（变量c）

在鲲鹏编译后生成指令

汇编代码	指令	说明
ldr x1, [sp,#12]	b9400fe1	从内存将变量a的值放入寄存器x1
ldr x0, [sp,#8]	b9400be0	从内存将变量b的值放入寄存器x0
add x0, x1, x0	0b000020	将x1(a)中的值加上x0(b)的值放入x0寄存器
str x0, [sp,#4]	b90007e0	将x0寄存器的值存入内存（变量c）

目录

迁移过程概述——五个步骤完成软件迁移

迁移准备

- 信息收集
- 环境申请

迁移分析

- 软件栈分析
- 编程语言/代码/依赖库分析

编译迁移

- 代码迁移
- 软件包迁移

性能调优

- 性能指标测试
- 性能优化

测试与认证

- 压力测试
- 长稳测试
- 规模商用/鲲鹏展翅认证

迁移准备——收集软件栈信息，准备迁移环境

信息收集

收集软件技术栈

软件信息

- 自研软件
- 开源软件
- 商业软件
- 中间件/编译器
- 操作系统/虚拟机

硬件信息

- 芯片/服务器信息

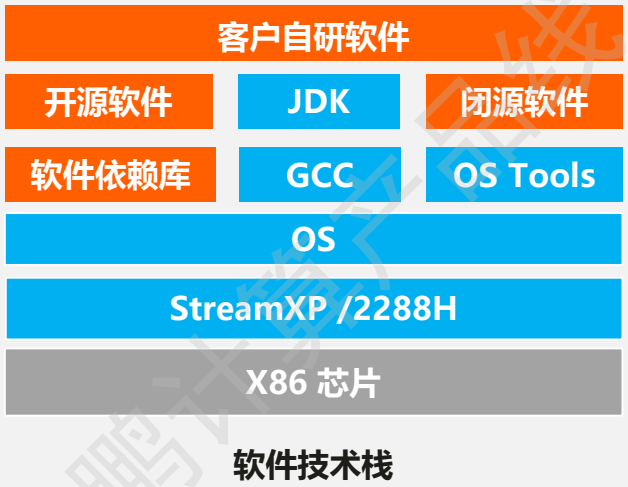
环境申请

Openlab远程环境，一站式迁移调优

业务流	源码移植检查	编译构建	性能调优	自动化测试	认证发布
测试套	兼容性测试	稳定性测试	性能测试	安全测试	功耗测试
工具链	移植知识库	扫描工具	移植工具	性能分析工具	自动化测试工具

- 办公环境可以连接公网环境
- 无需配置物理服务器，向Openlab申请远程服务器资源
- 提供认证，生态推广

迁移分析——分析软件栈，制定迁移策略



软件技术栈

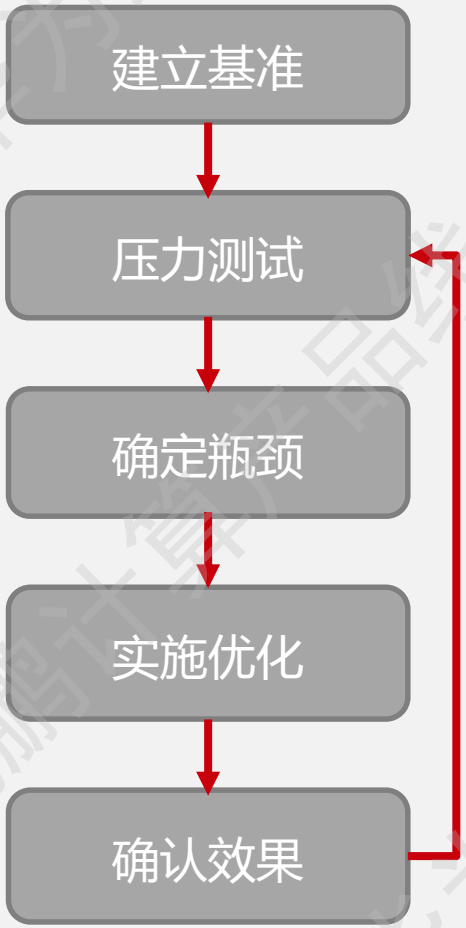
业务软件	开源软件	• **开源软件**：获取开源软件ARM64软件包或下载源码重新编译 • **依赖组件**：获取ARM64包或获取源码重新编译，闭源组件需升级替换ARM版本或替换类似组件库
业务软件	自研软件	• **编译语言**：C/C++/Go类编译语言重新编译 • **解释语言**：Java/Python等解释型语言替换ARM版本JDK或PVM虚拟机
业务软件	商用软件	• 获取支持鲲鹏处理器或TaiShan服务器的软件版本 • 如无法获取到兼容版本，需更换其他类似软件
运行环境	JDK	• 替换支持鲲鹏版本JDK，如OpenJDK
运行环境	编译器	• 使用操作系统自带的编译器或者下载源码重新编译，如GCC 7.3
运行环境	操作系统	• 替换支持鲲鹏操作系统，可使用智能计算产品兼容性查询助手查询

**说明：在华为云社区的鲲鹏论坛，已经建立了软件仓库，部分软件包可以在软件仓库直接下载*

编译迁移——软件编译打包，验证基本功能

迁移类型	子类	修改点
代码迁移	编译型语言 C/C++/Go	• **代码修改**：源码中对指令集依赖的**宏定义**及**builtin**函数 • **脚本修改**：替换成支持鲲鹏的**编译选项** • **内联汇编**：**替换后重新编译**，比如rdtsc指令等 • **不兼容指令**：如**SSE Intrinsic类加速指令**，需重新编写
	解释型语言 Java/Python	• **直接翻译**：纯解释型语言开发的应用程序，代码无需修改，程序也不需要重新编译， 如Java软件**安装ARM版本JDK即可** • **依赖库编译**：如果软件含依赖库，则**需要重新编译**
软件包迁移	RPM	• **扫描依赖**：扫描软件包的运行依赖库/可执行程序 • **依赖库编译**：编译型语言需**重新编译**，解释型语言基于对应的虚拟机运行 • **软件打包**：按照软件安装包原有结构重新制作成鲲鹏的软件包
	DEB	

性能调优——利用五步法优化软件性能



调优前根据当前的硬件配置、组网、测试模型做综合评估，建立合理的调优目标。

通过压力测试工具对系统加压，同时监控系统数据，记录数据变化。

在压力测试下，测试系统瓶颈比较容易显现。系统的瓶颈通常会在CPU过于繁忙、IO等待、网络等待、响应时延等方面出现。

根据瓶颈点针对性地实施优化措施，同时记录优化措施，一旦出现负向效果，及时回退。

重新启动压力测试，准备好相关的工具监视系统，确认优化效果。

测试与认证——保障商用上线，共建鲲鹏生态

层层测试，保障业务高性能稳定运行

功能测试

测试业务基本功能在鲲鹏上运行是否正常，含单元测试及系统集成测试

性能测试

- 测试业务性能表现是否符合目标，同时监测各系统指标是否正常

长稳测试

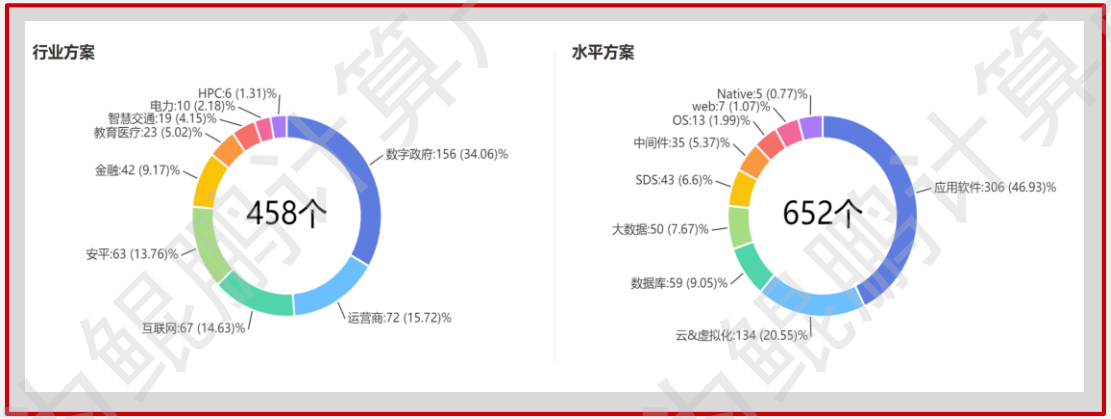
- 长时间运行测试程序，以检测业务能否长期稳定运行

规模商用

- 上市资料刷新
- 割接上线

鲲鹏展翅认证现状

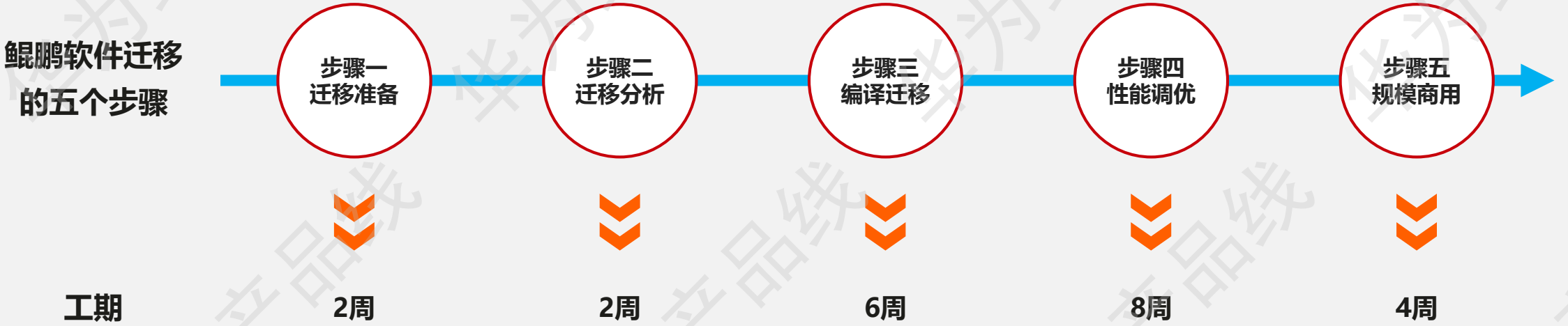
- 累计**909**家行业合作伙伴获得鲲鹏展翅认证，平均每月认证申请**63**份、发放证书**55**份。
- 覆盖数字政府、电力、安平、运营商、金融等9大**行业方案**，以及数据库、中间件、大数据、分布式存储、云&虚拟化等9个**水平方案**。



说明：上述数据为截止到2020年7月10日的统计数据

目录

华为内部项目：大型平台软件历时4个月完成规模商用



业务特征

- **代码规模**：C/C++（ **272万**行）、Java（ **165万**行 ）、Python（ **17万**行 ）和JavaScript（ **10.8万**行 ）
- **依赖库**：Java及C++的依赖库总计**140**个，其中C/C++语言相关的**34**个
- **性能目标**：提升10%

迁移修改

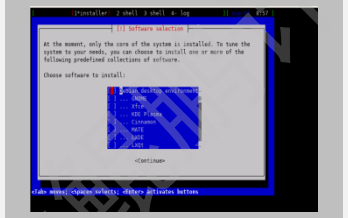
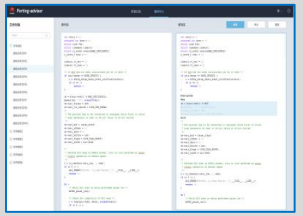
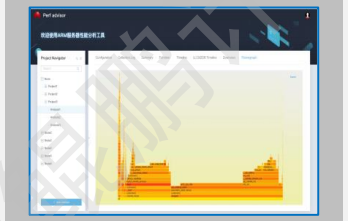
- 174个依赖库需要重新编译
- 代码归一，构建脚本归一
- 汇编指令替换
- 编译选项修改
- 数据类型修改

互联网行业伙伴核心应用迁移

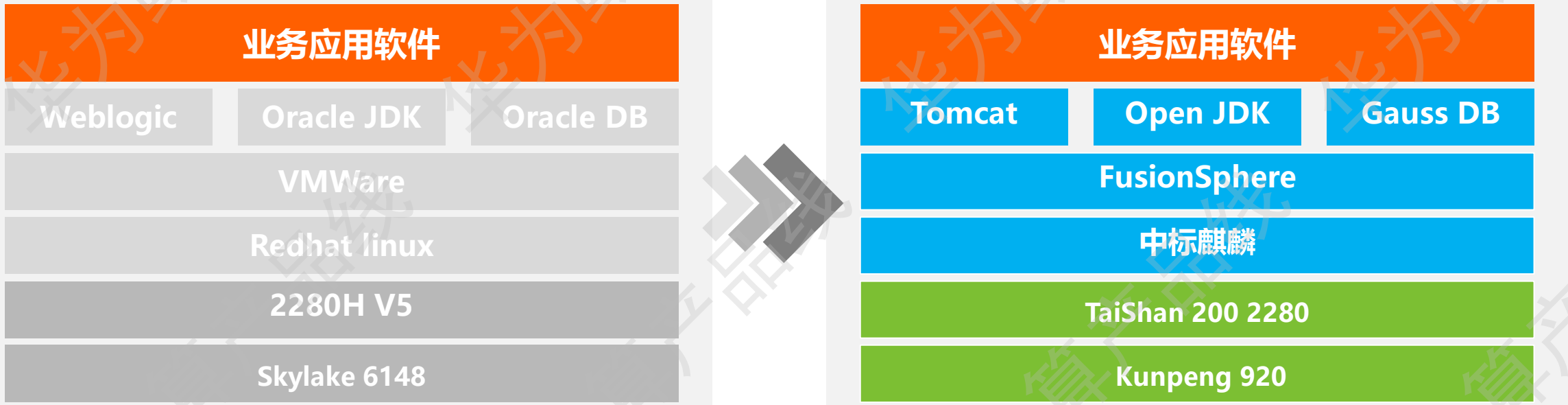
案例：行业伙伴核心应用系统迁移

业务系统 C++/汇编/Java 520+源码文件	→	**业务系统** 55个依赖库移植 39个编译问题，195行代码修改
Debian OS	→	**Debian OS** 适配补丁（30个）
原有平台 2.4GHz / 2P / 48核	→	**鲲鹏/TaiShan** 2.6GHz / 2P / 96核

5人月完成520+源码文件移植，性能提升10%

环境准备：1人月	代码编译和修改：2人月	性能优化：2人月
技术分析 硬件环境 代码评估 环境部署	**代码移植** C++/汇编语言移植 安装JDK 编译开源代码 更换依赖库	**性能优化** 热点函数分析 多核绑定 内存就近访问 网卡队列优化
		

金融行业核心系统整体切换案例



- **迁移规模**：整体迁移，涉及的商业软件较多，编码类型主要以Java居多，涉及的Java软件包100多个
- **测试场景**：大数据/数据库/虚拟化/物理机性能/业务系统功能
- **整体用时**：整体迁移用时1.5个月，测试及调优用时1个月
- **测试结果**：各业务系统功能验证通过，各测试场景性能均优于友商

鲲鹏开发套件，使能开发者，加速算力升级

迁移分析
技术可行性

编译迁移
功能可用

性能调优
性能更优

Hello Kunpeng!

分析扫描工具

- 软件兼容性列表
- 代码评估
- 依赖库检查评估
- 改动量评估

代码迁移工具

- 软件构建迁移
- 源文件迁移
- 依赖库迁移
- 移植指导报告

性能优化工具

- 性能分析全景
- 系统/进程/线程函数分析
- 软件调优建议
- 性能分析&调优

目录

HUAWEI

前言

本章讲解了C/C++代码编译原理及构建流程，针对其中典型移植类问题进行分析讲解，重点介绍各类移植问题的迁移方法。

目标

学完本章节后，您将能够学习到：

- C/C++代码编译原理、编译构建流程
- C/C++代码迁移过程中可能涉及的移植项
- 典型迁移类问题移植方法

目录

从源码到可执行程序——编译型语言

- **编译型语言**：典型的如C/C++/Go语言，都属于编译型语言。编译型语言开发的程序在从x86处理器迁移到鲲鹏处理器时，必须经过重新编译才能运行。
- 从源码到程序的过程：源码需要由编译器、汇编器翻译成机器指令，再通过链接器链接库函数生成机器语言程序。

C/C++代码需移植的原因：

- **架构差异**：X86和鲲鹏处理器(aarch64)属于不同的架构。
- **指令集差异**：X86--复杂指令集，鲲鹏处理器--精简指令集。
- **向量寄存器差异**：X86和鲲鹏处理器使用向量寄存器不同，向量指令集也存在差异。

目录

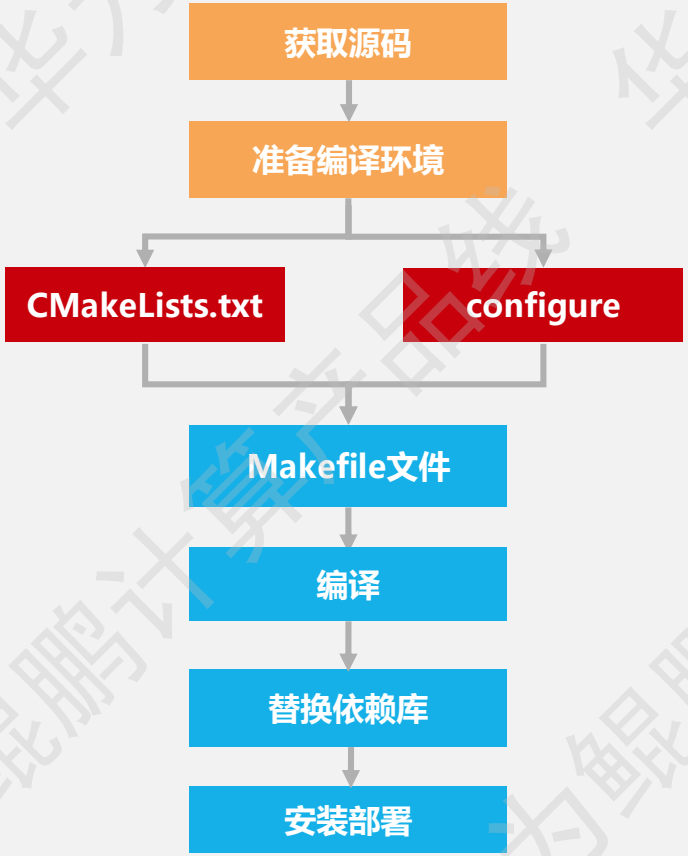
C/C++代码编译构建过程

C/C++代码工程主要包括两类文件：编译构建脚本、C/C++源码

可能涉及的移植项：

- **编译构建脚本类文件**
  - 编译选项的移植（指定数据类型、生成代码特性、目标执行器架构、处理器硬件加速功能等）
- **C/C++源码类文件**
  - 编译宏移植（用户自定义宏移植、编译器自定义宏移植）
  - 编译器自带builtin函数移植
  - 内联汇编移植
  - SSE intrinsic函数移植（MMX/SSE类函数移植、AVX函数移植）

C/C++代码编译构建过程



1、获取源码：通过github或第三方开源社区获取

2、准备编译环境：安装编译器gcc等

3、使用源码中的CMakeLists.txt或configure脚本生成makefile

4、执行makefile编译可执行程序

5、替换依赖库：重编译或替换依赖X86平台的so库

6、将可执行程序安装部署到生产或测试系统

**说明：configure脚本也常由源码中的autogen.sh或bootstrap.sh脚本执行后产生*

目录

代码迁移—编译脚本、编译选项移植

	功能说明	X86代码	鲲鹏代码
64位编译	定义编译生成的应用程序为64位	-m64	-mabi=lp64
数据类型	显示定义char类型变量为有符号类型	char a = ‘a’	方法1：signed char a = ‘a’ 方法2：-fsigned-char

编译选项修改(典型编译脚本修改参考)：

- Cmakelists.txt文件修改
  - add_definitions(-Wall -mabi=lp64 -g)
  - set(CMAKE_CXX_FLAGS “-Wall -mabi=lp64 –g”)
  - add_compile_options(-Wall -mabi=lp64 -g)
- Makefile文件修改

cc/g++ -Wall -mabi=lp64 -g –o …..

不同架构下差异化GCC编译选项查询 (gcc7.3为例)

链接：https://gcc.gnu.org/onlinedocs/gcc-7.3.0/gcc/Submodel-Options.html#Submodel-Options

代码迁移—编译宏移植

	功能	X86编译选项	鲲鹏编译选项
编译宏	原有X86编译宏移植 (gcc编译器自定义宏)	__x86_64__或__x86_64	__aarch64__
	平台属性意义编译宏移植 (用户自定义编译宏)	HAVE_X86_64	HAVE_AARCH64

功能说明	X86代码(编译器自定义宏)	鲲鹏代码
说明平台属性的宏定义	__amd64__或__amd64	__aarch64__
SIMD属性相关宏	__SSE__ /__SSE2__/__SSE3__ /__SSE4_1__/__SSE4_2__/	自定义NEON语义编译宏，实现对应功能分支

注：使用 gcc -march=x86-64 -dM -E - < /dev/null 查看编译器自定义宏(鲲鹏下-march=armv8-a)

代码迁移—builtin函数移植

	功能	X86指令	鲲鹏指令
Builtin函数	8bit数据的crc32校验值计算	__builtin_ia32_crc32qi (__a, __b)	__builtin_aarch64_crc32cb (__a, __b)

需移植的普通builtin函数实际并不多，大部分需移植的builtin函数集中在 SSE intrinsic函数内，这部分的移植将在后续详细讲解。

功能说明	X86代码	鲲鹏代码
计算16bit数的crc32校验值	__builtin_ia32_crc32hi (__a, __b)	__builtin_aarch64_crc32ch (__a, _b)
计算32bit数的crc32校验值	__builtin_ia32_crc32si (__a, __b)	__builtin_aarch64_crc32cw (__a, _b)
计算64bit数的crc32校验值	__builtin_ia32_crc32di (__a, __b)	__builtin_aarch64_crc32cx (__a, _b)

注：SSE intrinsic函数也是基于builtin函数进行封装的，这类builtin函数移植占有较大比重，后续将详细介绍。

代码迁移—内联汇编函数移植

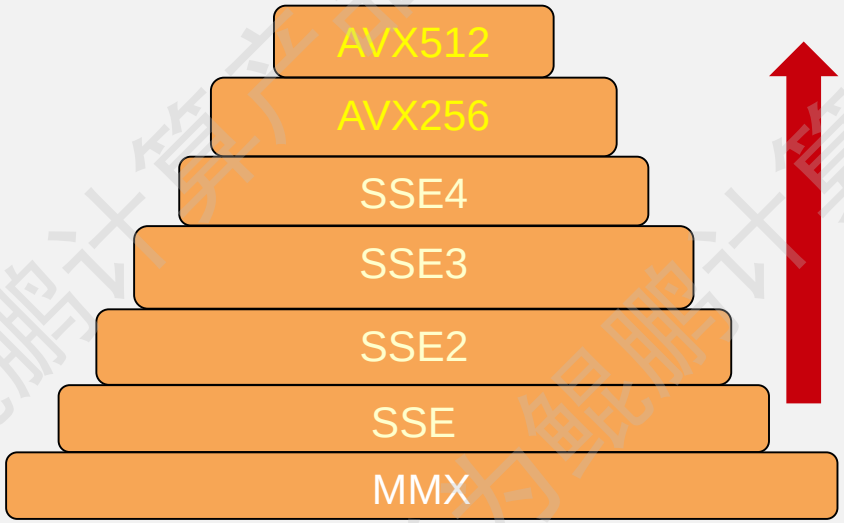
	功能	X86指令	鲲鹏指令
内联汇编	将字节序进行反序 (汇编指令方式替换) Val: 0x56781314 => 0x14137856	__asm__("bswap %0" : "=r" (val) : "0" (val))	__asm__("rev %w[dst], %w[src]" : [dst]"=r"(val) : [src]"r"(val))
	计算变量a（uint64）的二进制中1的个数 (builtin函数方式替换) a: 0111 (7) => result: 3	__asm__("popcnt %1, %0" : "=r"(result) : "mr"(a) : "cc")	Result = __builtin_popcountll（a）

内联汇编规则参考：https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html#Using-Assembly-Language-with-C

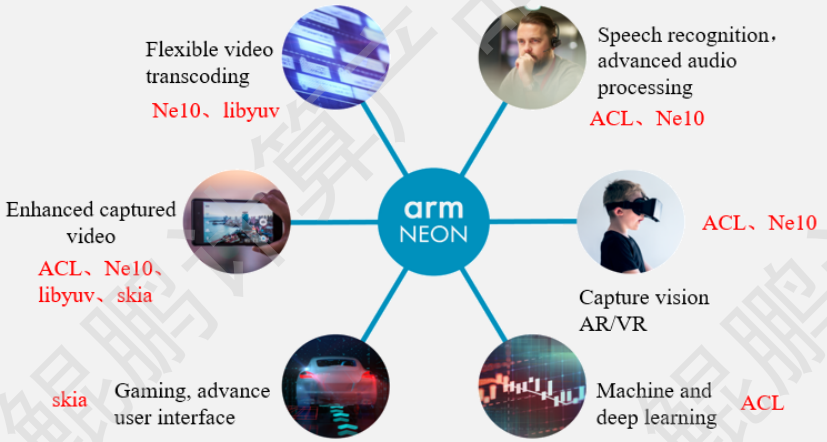
代码迁移—SSE intrinsic函数移植(SIMD技术简介)

- SIMD(Single Instruction Multi Data)是一种单指令处理多数据流的并行处理技术，能够在批量数据操作时进行向量化运算加速，具有较高的执行效率，在多媒体处理、矩阵运算等场景都有广泛的应用。

- SSE (Intel的SIMD扩展指令集的简称)



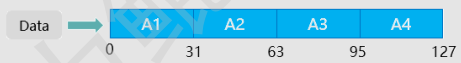
- NEON（基于SIMD思想的ARM技术）



图片来源arm neon官网主页

典型NEON Lib库：Arm Compute Library(ACL)、Ne10、libyuv、skia

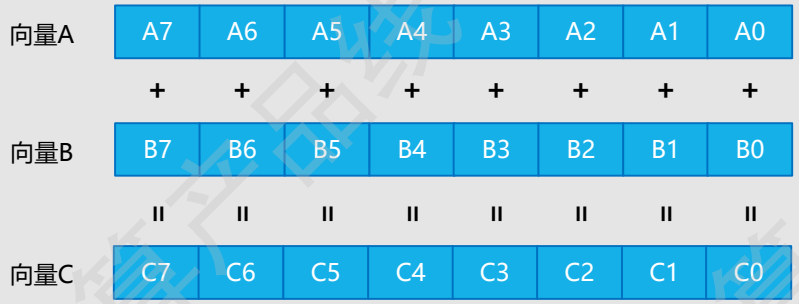
代码迁移—SSE intrinsic函数移植(MMX/SSE)

	功能	X86指令	鲲鹏指令
MMX指令	将向量a和向量b中的（int32）元素进行向量加法运算 	__m64 _mm_add_pi32 (__m64 a, __m64 b)	int32x2_t vadd_s32 (int32x2_t __a, int32x2_t __b)
SSE指令	从内存（p地址）加载4个单精度浮点数据到寄存器 	__m128 _mm_load_ps (float *p)	float32x4_t vld1q_f32(float *p)

代码迁移—SSE intrinsic函数移植（AVX）

AVX指令计算16个单精度浮点数之和

__m256d _mm256_add_ps (__m256 A, __m256 B)



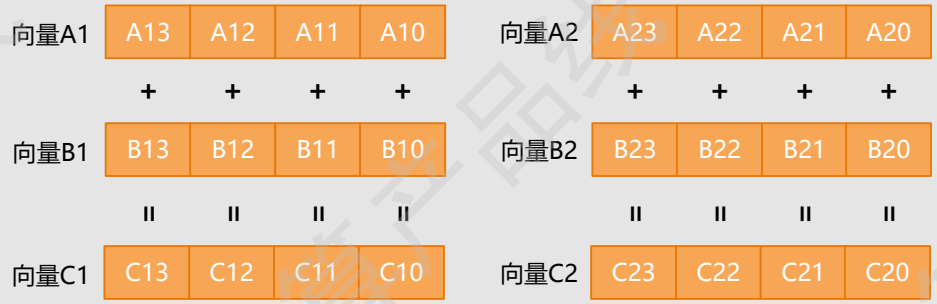
D = C7 + C6 + C5 + C4 + C3 + C2 + C1 + C0

AVX指令说明：

- 本例中AVX指令使用了256位寄存器运算，向量A和向量B中分别存储了8个单精度浮点型（32位）。
- 该指令将向量A和向量B中的8个数值分别相加，并将结果以返回值的形式返回（结果中依然是8个单精度浮点型数据）
- 最后从向量寄存器中分别取出8个单精度浮点数累加得到结果

鲲鹏指令计算16个单精度浮点数之和

float32x4_t C1=vaddq_f32(A1, B1)
float32x4_t C2=vaddq_f32(A2, B2)



D = C13 + C12 + C11 + C10 + C23 + C22 + C21 + C20

鲲鹏指令说明：

- 鲲鹏处理器采用精简指令集，使用128位寄存器实现SIMD（Single Instruction Multi Data）计算。
- 在实现本例16个浮点数的相加时，使用两条vaddq_f32指令分别完成，每条指令完成两组共8个浮点数计算
- 最后再从向量寄存器中分别取出8个浮点数累加

代码迁移—SSE intrinsic函数移植方法

- **方法1：基于avx2neon.h、SSE2NEON.h开源文件移植**
  - 鲲鹏AvxToNeon开源工程： https://github.com/kunpengcompute/AvxToNeon
    - 包含SSE类intrinsic函数的NEON实现（字符串比较、crc32值计算、popcnt计算等）
    - 包含基础的AVX256、AVX512类intrinsic函数的NEON实现（load、store、运算、移位等操作指令）
  - 开源的SSE2NEON工程：https://github.com/DLTcollab/sse2neon/blob/master/sse2neon.h
    - 主要实现SSE类intrinsic函数替换(整数、单浮点数据类型)
    - 涵盖基础的load、store、set、运算操作等指令的NEON实现

- **方法2：手动替换移植**
  - SSE Intrinsics Guide网站：https://software.intel.com/sites/landingpage/IntrinsicsGuide/
  - NEON Intrinsic Guide网站： https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics
  - 在需移植文件中添加头文件 arm_neon.h

目录

迁移工具—Porting Advisor初步代码扫描

- Porting Advisor是一款华为鲲鹏代码迁移工具，针对C/C++代码进行扫描分析，检查用户C/C++代码中需移植修改的MakeFile编译文件、X86汇编及SSE intrinsic函数，并指导用户如何移植。
- **鲲鹏开发套件 — Porting Advisor:**
  https://www.huaweicloud.com/kunpeng/software/portingadvisor.html

迁移工具—Porting Advisor初步代码扫描

- 以大数据中的Impala组件为例

1.Impala源码下载及扫描分析

2.扫描报告解读及运用

Source files scan details are as follows:

filename	filetype	lineno	rows	category	keyword	suggestion	description

Impala-cdh	FileType.C	(83, 83)	1	PortingCategory.BuiltinAssembles	RDTSC	static uint64_t Rdtsc() {
						uint64_t count_num;
						Current_Speed = 2400; // Current Speed =2400MHz
						External_Clock = 100; // External Clock = 100MHz
						__asm__ __volatile__ ("mrs %0, cntvct_el0" : "=r" (count_num));
						return count_num * (Current_Speed / External_Clock); }
							__asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi))

Impala-cdh	FileType.C	(139, 139	1	PortingCategory.Intrinsics	_mm_set_epi8	Please update the source code and add '#include "sse2neon.h"' on the head of the file.You could download the file sse2neon.h from https://github.com/DLTcollab/sse2neon/blob/master/sse2neon.h.	Set packed 8bit integers.
Impala-cdh	FileType.C	(271, 271	1	PortingCategory.BuiltinAssembles	PAUSE	These assembles need to port.	Found arch specific assembles using '__asm'/'__asm__'
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

本章总结

- 本章简要介绍了C/C++代码的编译原理及构建流程，重点对典型迁移类问题及方法进行详细介绍。最后引入Porting Advisor开发工具，帮助学员快速、高效的完成C/C++代码迁移。

- 核心类移植项及迁移方法：
  - ✓ **编译选项移植**：关注平台差异项（用好编译器官方文档）
  - ✓ **编译宏移植**：区分编译器/用户自定义宏移植
  - ✓ **builtin函数移植**：关注常用builtin函数替换
  - ✓ **内联汇编移植**：识别核心汇编指令，用好汇编指令/builtin函数替换
  - ✓ **SSE intrinsic函数移植**: 开源工程、官网指导手册相结合

目录

HUAWEI

前言

本章节主要介绍Java/Python代码编译运行过程涉及的迁移改动点，及针对迁移改动点的处理方法。

目标

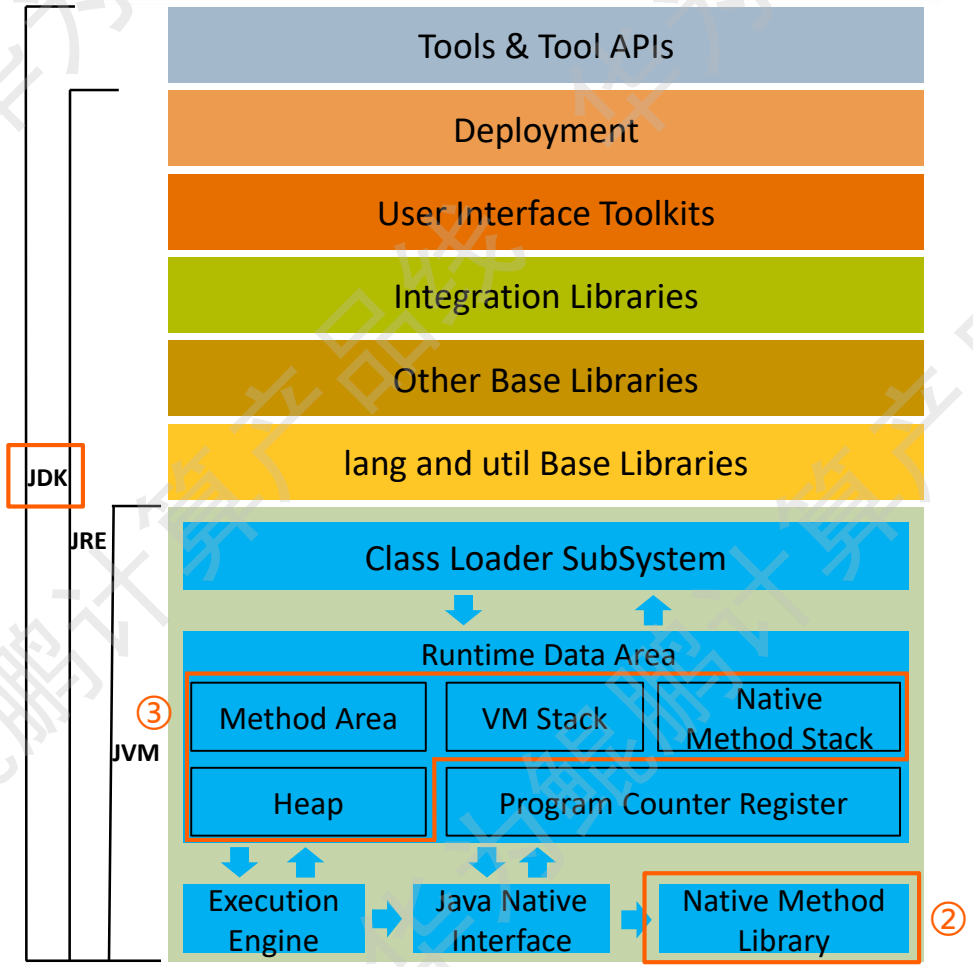
学完本课程后，您将能够：

- 了解Java/Python的运行过程中可能涉及的迁移改动点
- 掌握典型场景的迁移方法
- 独立完成简单Java/Python代码的迁移

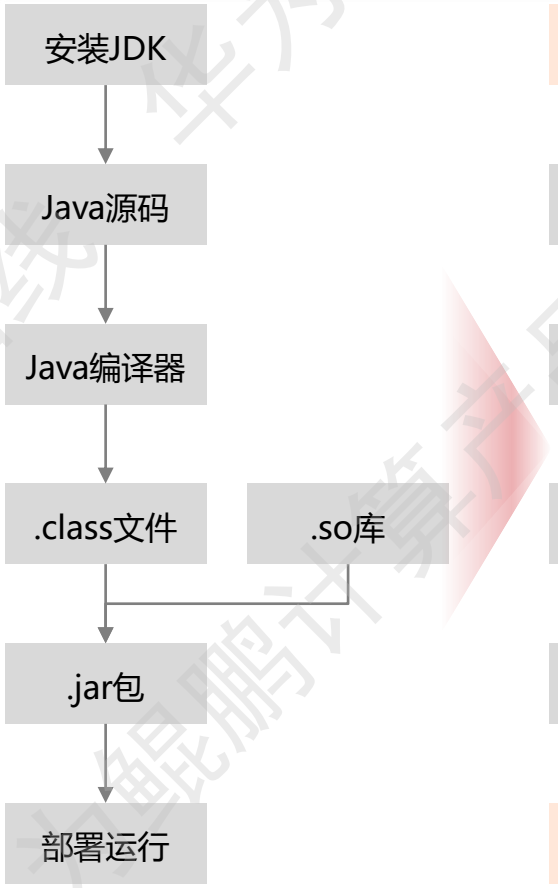
目录

Java从源码到可执行程序

Java技术架构



Java源码迁移修改点



① 安装JDK改动点：

- 定制版本需源码编译

② 引用SO库的改动点：

- 替换成aarch64的SO库

③ 程序运行时改动点：

- 设置JVM参数

Java代码迁移：安装合适的JDK版本

升级成已适配的JDK

为什么要用稳定成熟的高JDK版本?

- 生产环境更加注重稳定性
- 对老版本的问题进行了修复和改进
- 新增特性使编程更加方便简洁
- OpenJDK适配，支持通过yum命令直接安装

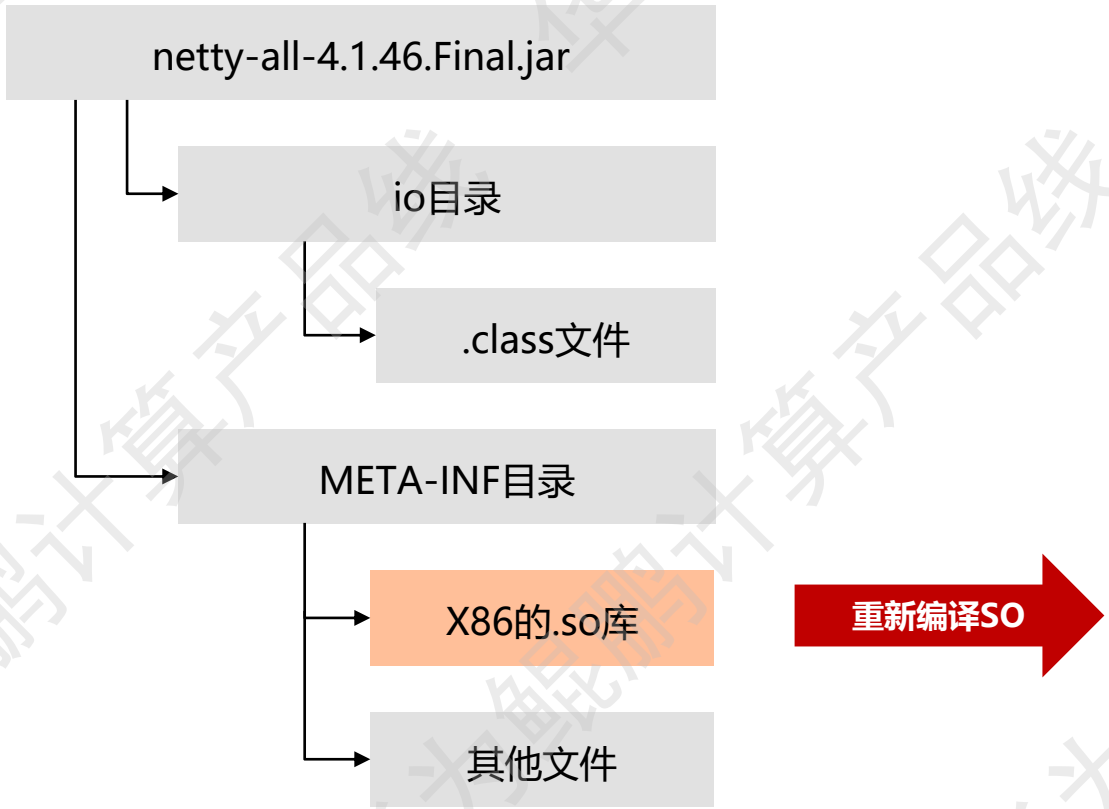
需要源码编译、部署的JDK

源码编译、安装定制的JDK或其他版本JDK

- 安装GCC
- 获取JDK源码
- 配置编译选项(部分列举)
  - ① export LANG=C # [必选]语言选项
  - ② --with-target-bits=64 # 64位机器
  - ③ --disable-warnings-as-errors # 忽略warning
  - ④ --with-debuglevel=slowdebug #调试等级
- make all启动编译
- 编译完成，设置环境变量
- 进行验证(java -version)

Java代码迁移：包含SO库调用的jar包迁移方法

Jar包结构，调用了SO库



重新编译SO库替换，完成迁移

迁移步骤：

- Dependency Advisor工具分析扫描jar包
- 识别依赖SO库
- 下载SO库源码
- 安装maven
- 安装GCC
- 设置参数-fsigned-char
- 编译aarch64版本SO库
- 替换SO库
- 重新打包jar文件

Java代码迁移：设置JVM参数，稳定快速的运行程序

JVM参数设置：3个经验+2个差异

经验1：完成一次Full GC后，应该释放出70%的堆空间（30%的空间仍然占用）。

经验2：假设老年代存活对象(即Full GC后老年代内存占用) 大小为X。

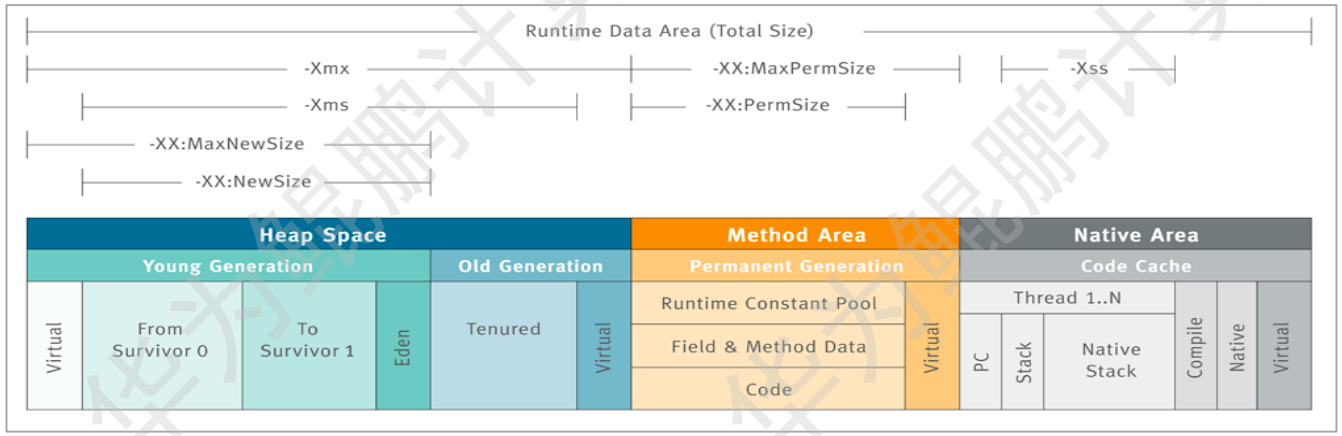
空间	倍数	参数
堆总大小	X的3-4倍	Xmx，Xms
年轻代	X的1-1.5倍	Xmn(NewSize+MaxNewSize)
老年代	X的2-3倍	堆和年轻代相减(Xmx-Xmn)
永久代	X的1.2-1.5倍	PermSize，MaxPermSize

经验3：官方建议年轻代大小占整个堆空间大小的3/8左右。

差异1：线程栈大小的Xss参数在ARM上默认值为2M，在X86上为1M，若线程开的太多，需要使用Xss调整线程栈的大小，防止OOM

差异2：由于ARM和X86指令集的差异性，导致JDK的JIT编译存在差异。可以通过设置参数ReservedCodeCacheSize，调整CodeCache大小。

参数控制区间的直观展示



案例分享：源码编译、安装OpenJDK9

1、下载JDK源码

- yum install mercurial -y
- hg clone http://hg.openjdk.java.net/jdk9/jdk9
- cd jdk9
- sh get_source.sh

JDK源码目录结构

```
[root@localhost jdk9_bak]# ls
ASSEMBLY_EXCEPTION  configure  get_source.sh  jaxp   jdk        LICENSE  Makefile  README
common              corba      hotspot        jaxws  langtools  make     nashorn   test
```

2、设置参数，开始编译

- 安装依赖：yum groupinstall "Development Tools" java-1.8.0-openjdk-devel –y
- 配置参数：bash configure --with-target-bits=64 --disable-warnings-as-errors --with-debuglevel=slowdebug --with-boot-jdk=/opt/tools/installed/jdk8u191-b12
- 编译：make all
- 验证：build/linux-aarch64-normal-server-slowdebug/jdk/bin/java -version

```
[root@localhost jdk]# ./bin/java -version
openjdk version "9-internal"
OpenJDK Runtime Environment (slowdebug build
OpenJDK 64-Bit Server VM (slowdebug build 9-
```

案例分享：netty-all-4.1.34.Final.jar迁移

1、分析扫描工具(Dependency Advisor)扫描结果，有1个依赖SO库需要移植

软件扫描信息配置

软件安装包存放路径或安装包名称	/opt/depadv/depadmin/netty-all-4.1.34.Final.jar	编译器版本	
		构建工具	
软件已安装路径		编译命令	
源代码存放路径			

目标环境配置

目标操作系统	CentOS 7.6
目标系统内核版本	4.14.0

分析结果

扫描出需要移植的SO库

依赖库文件　　总数：1，需要迁移：1

序号	名称	路径	处理建议	操作
1	netty-all-4.1.34.Final.j...	/netty-all-4.1.34.Final.jar	jar包在鲲鹏平台已存在兼容版本，URL为jar包下载地址。	下载
		/META-INF/native/libnetty_transport_native_epoll_x86_64.so		

需要迁移的源文件　　0

需要迁移的代码行数　　C/C++和Makefile源代码：0行；汇编代码：0行

预估迁移工作量

预估标准：1人月迁移工作量 = 500行C/C++源代码，或250行汇编代码

工作量：0 人月

案例分享：netty-all-4.1.34.Final.jar迁移

2、将jar包中调用的.so库重新编译替换为aarch64版本，并重新打jar包(mvn自动完成)

- 安装GCC，设置参数-fsigned-char
- 安装maven
- 安装OpenJDK
- 下载netty-all-4.1.34和依赖netty-tcnative-2.0.22
- 解压netty-tcnative-2.0.22，并编译(mvn install)
- 解压netty-all-4.1.34，并编译(mvn install)
- 编译生成的jar包解压后确认已经包含aarch64的SO库

```
/data/netty-test2/netty-netty-4.1.34.Final/all/target
total 6752
drwxr-xr-x. 2 root root    4096 Mar  3 19:21 antrun
drwxr-xr-x. 2 root root    4096 Mar  3 19:21 dependency-maven-plugin-markers
drwxr-xr-x. 3 root root    4096 Mar  3 19:21 dev-tools
drwxr-xr-x. 2 root root    4096 Mar  3 19:21 maven-archiver
-rw-r--r--. 1 root root 3064465 Mar  3 19:21 netty-all-4.1.34.Final-sources.jar
-rw-r--r--. 1 root root 3826848 Mar  3 19:21 netty-all-4.1.34.Final.jar
```

jar解压

```
[root@localhost target]# tree netty-all-4.1.34.Final/META-INF/
netty-all-4.1.34.Final/META-INF/
|-- INDEX.LIST
|-- MANIFEST.MF
|-- io.netty.versions.properties
|-- maven
|   `-- io.netty
|       `-- netty-all
|           |-- pom.properties
|           `-- pom.xml
`-- native
    `-- libnetty_transport_native_epoll_aarch_64.so
```

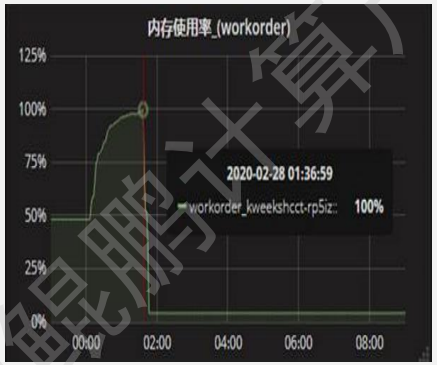
案例分享：JVM参数设置，解决服务挂死问题

1、调整线程栈大小或内存大小，解决OOM问题

问题描述：

- 鲲鹏容器压测8小时，在第3个小时有4台鲲鹏容器内存全部耗尽，服务主进程挂死

问题现象



1、排除业务代码内存泄漏

措施：

将ARM容器内存大小从4G提升为8G，长稳测试

结果：

4台ARM容器正常运行未挂死

结论：

非业务代码导致的内存泄露

2、分析ARM和X86上JVM参数差异

措施：

全面对比jvm运行参数版本信息

结果：

- JVM版本一致，堆空间大小一致
- 线程栈大小不一致，ARM为2M，X86为1M

结论：

线程栈多1M，会导致450个线程多用450M内存，超过堆内存，引发OOM

3、总结

- ARM服务器上默认的线程堆栈大，线程多会导致比X86多占用内存，触发OOM
- 建议修改参数减少栈大小(-Xss1m)，或者扩充内存(-Xmx8g -Xms8g)。

Java迁移小结

一、安装JDK版本

- 通过yum安装
- 源码编译安装

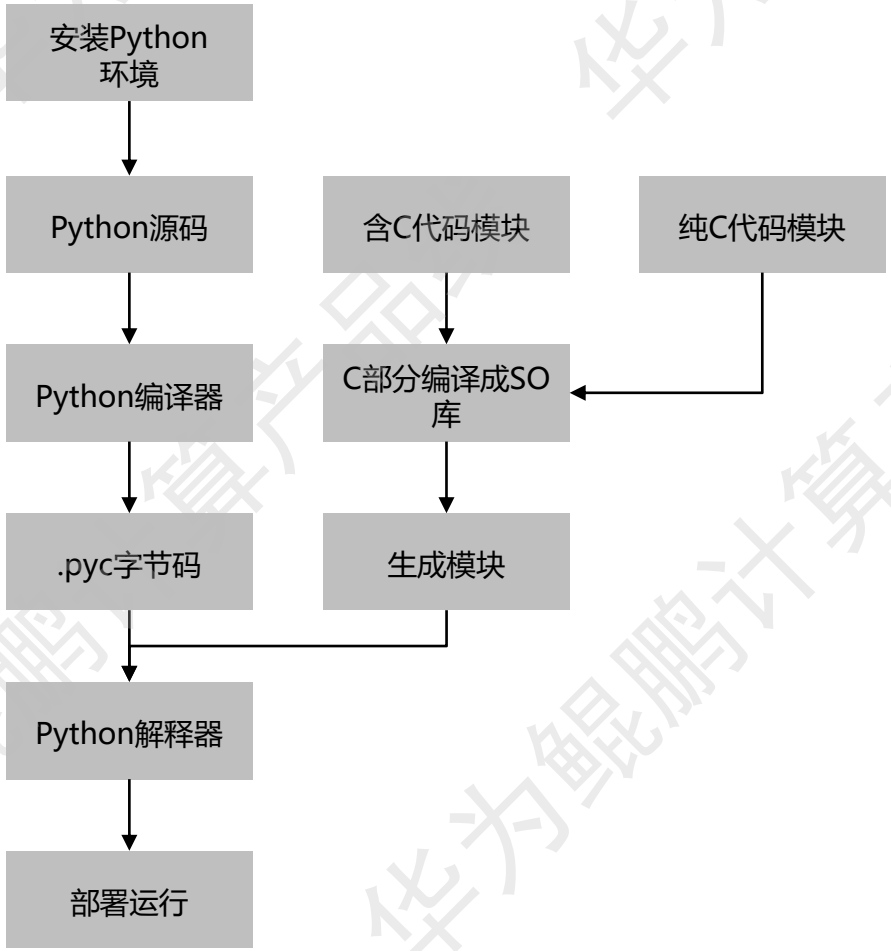
二、引用的SO库需重新编译

三、根据业务实际情况，调整JVM参数

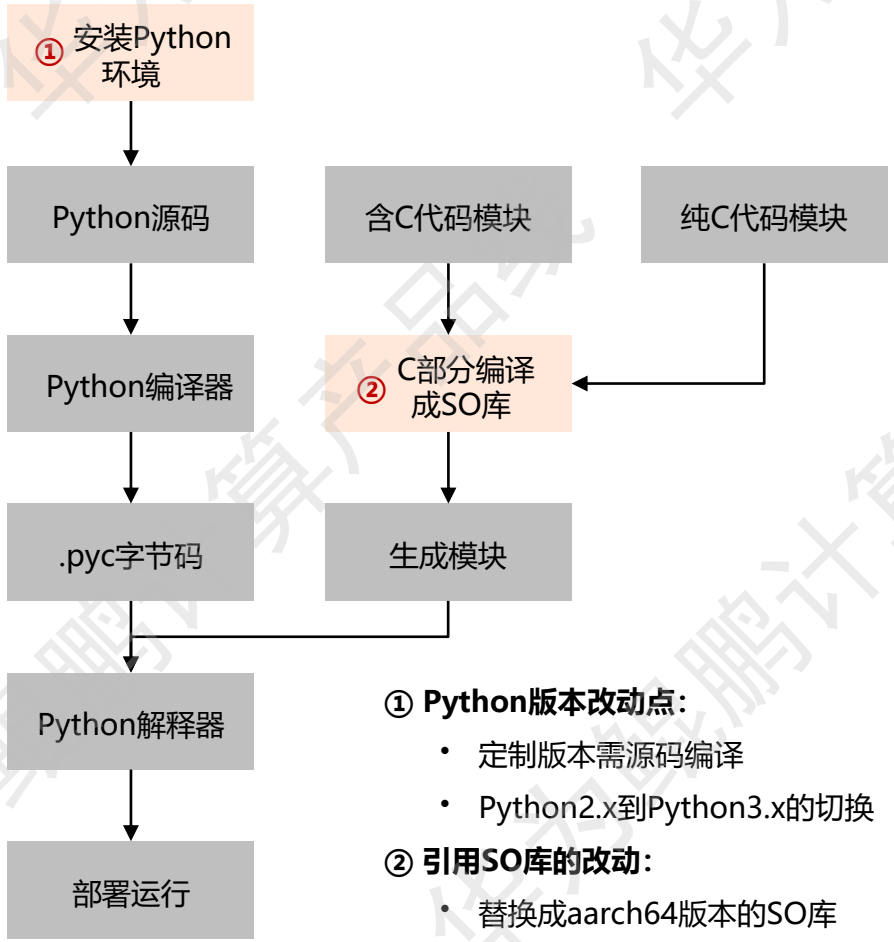
目录

Python从源码到可执行程序

Python从源码到运行



Python代码迁移改动点



① Python版本改动点：

- 定制版本需源码编译
- Python2.x到Python3.x的切换

② 引用SO库的改动：

- 替换成aarch64版本的SO库

Python代码迁移：升级Python版本

（推荐）升级Python3.X版本	源码编译安装Python3.X版本
yum install python3 -y	• 安装GCC，配置编译选项-fsigned-char • 官网下载安装包：wget https://www.python.org/ftp/python/3.8.2/Python-3.8.2.tgz • 解压源码包：tar zxvf Python-3.8.2.tgz && cd Python-3.8.2 • 配置编译选项：./configure --prefix=/usr/python3.8 • 编译安装：make && make install • 验证：/usr/python3.8/bin/python3

```
[root@localhost Python-3.8.2]# /usr/local/python3.8/bin/python3
Python 3.8.2 (default, Mar  5 2020, 02:23:29)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python代码迁移：含C模块或全C模块的迁移

含C代码编写的模块，Python C API调用SO库

```
[root@localhost site-packages]# tree numpy/ | grep so
    ├── _multiarray_tests.cpython-36m-x86_64-linux-gnu.so
    ├── _multiarray_umath.cpython-36m-x86_64-linux-gnu.so
    ├── _operand_flag_tests.cpython-36m-x86_64-linux-gnu.so
    ├── _rational_tests.cpython-36m-x86_64-linux-gnu.so
    ├── _struct_ufunc_tests.cpython-36m-x86_64-linux-gnu.so
    ├── _umath_tests.cpython-36m-x86_64-linux-gnu.so
    ├── absoft.py
    │   ├── absoft.cpython-36.pyc
    ├── _pocketfft_internal.cpython-36m-x86_64-linux-gnu.so
    ├── _datasource.py
    │   ├── _datasource.cpython-36.pyc
    │   ├── test__datasource.cpython-36.pyc
    │   ├── test__datasource.py
    ├── lapack_lite.cpython-36m-x86_64-linux-gnu.so
    ├── _umath_linalg.cpython-36m-x86_64-linux-gnu.so
    │   └── timer_comparison.cpython-36.pyc
    ├── timer_comparison.py
    ├── _bit_generator.cpython-36m-x86_64-linux-gnu.so
    ├── _bounded_integers.cpython-36m-x86_64-linux-gnu.so
    ├── _common.cpython-36m-x86_64-linux-gnu.so
    ├── _generator.cpython-36m-x86_64-linux-gnu.so
    ├── _mt19937.cpython-36m-x86_64-linux-gnu.so
    ├── mtrand.cpython-36m-x86_64-linux-gnu.so
    ├── _pcg64.cpython-36m-x86_64-linux-gnu.so
    ├── _philox.cpython-36m-x86_64-linux-gnu.so
    ├── _sfc64.cpython-36m-x86_64-linux-gnu.so
```

纯C代码编写的模块，上层包裹了一层Python接口

```
[root@localhost site-packages]# tree cv2/
cv2/
├── config-3.6.py
├── config.py
├── __init__.py
├── load_config_py2.py
├── load_config_py3.py
├── __pycache__
│   ├── __init__.cpython-36.pyc
│   └── load_config_py3.cpython-36.pyc
└── python-3.6
    └── cv2.cpython-36m-x86_64-linux-gnu.so

2 directories, 8 files
```

核心都是SO库的调用

重新编译SO

重新编译SO库替换，完成迁移

迁移步骤：

- Porting Advisor工具分析源码
- 识别依赖SO库，C代码
- 下载模块源码
- 安装GCC
- 配置-fsigned-char选项
- 执行setup.py自动完成模块编译
- 完成aarch64版本SO库的替换
- 编译的模块安装到site-packages目录下，供其他Python源码引用

Python案例分享：numpy-1.18.1模块迁移

1、分析扫描工具(Dependency Advisor)扫描numpy

软件扫描信息配置

软件安装包存放路径或安装包名称		编译器版本	
软件已安装路径	/opt/depadv/depadmin/numpy	构建工具	
源代码存放路径		编译命令	

目标环境配置

目标操作系统	CentOS 7.6
目标系统内核版本	4.14.0

分析结果

扫描出需要移植的SO库

依赖库文件　　总数：21，需要迁移：21

序号	名称	路径	处理建议	操作
1	libgfortran-ed201abd.so.3	/opt/depadv/depadmin/numpy/.libs/libgfortran-ed201abd.so.3.0.0	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认
2	_pcg64.cpython-36m-x86_64-linux-gnu.so	/opt/depadv/depadmin/numpy/random/_pcg64.cpython-36m-x86_64-linux-gnu...	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认
3	_generator.cpython-36m-x86_64-linux-gnu.so	/opt/depadv/depadmin/numpy/random/_generator.cpython-36m-x86_64-linux-...	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认
4	_umath_linalg.cpython-36m-x86_64-linux-gnu.so	/opt/depadv/depadmin/numpy/linalg/_umath_linalg.cpython-36m-x86_64-linux-...	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认
5	_multiarray_umath.cpython-36m-x86_64-linux-gnu.so	/opt/depadv/depadmin/numpy/core/_multiarray_umath.cpython-36m-x86_64-li...	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认
6	_pocketfft_internal.cpython-36m-x86_64-linux-gnu.so	/opt/depadv/depadmin/numpy/fft/_pocketfft_internal.cpython-36m-x86_64-linu...	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认
7	_bit_generator.cpython-36m-x86_64-linux-gnu.so	/opt/depadv/depadmin/numpy/random/_bit_generator.cpython-36m-x86_64-lin...	无法确认鲲鹏平台是否支持该文件，请检查确认	无法找到下载链接，请检查确认

需要迁移的源文件　　0

需要迁移的代码行数　　C/C++和Makefile源代码：0行；汇编代码：0行

预估迁移工作量

预估标准：1人月迁移工作量 = 500行C/C++源代码，或250行汇编代码

工作量：0 人月

Python案例分享：numpy-1.18.1模块迁移

2、重新编译aarch64版本SO，并替换

- numpy符合C99标准，所以需要设置C99编译: export CFLAGS= ‘-std=c99’
- 安装Cython >=0.29.2: pip install Cython（默认使用最新版本0.29.15）
- 编译模块: python3 setup.py build

3、安装替换SO库后的模块，并编码验证

- 安装numpy模块：python3 setup.py install
- 验证结果
  - 目录验证：安装成功后在site-packages会有以numpy开头的目录
  - 编码验证：

```
[root@localhost python-test]# python3
Python 3.8.2 (default, Mar  5 2020, 02:23:29)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from numpy import *
>>> eye(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
>>>
```

本章总结

一、2个解释型语言的移植通用处理方法：

- 解释器或虚拟机(JVM)本身的迁移处理方法：直接安装RPM或者源码编译
- 调用含SO库的jar包或模块，需要替换aarch64版本的SO库才能正常使用

二、Java代码可以分析源码的空间占用，预先设置JVM的调优参数

三、使用分析扫描工具可以加快代码迁移过程

目录

HUAWEI

前言

本章节主要介绍maven仓的分类，搜索顺序，及如何使用鲲鹏maven仓。

目标

学完本课程后，您将能够：

- 了解Maven仓分类和搜索顺序；
- 了解鲲鹏Maven仓解决的问题和下载路径；
- 熟悉如何配置优先搜索鲲鹏Maven仓；
- Hive编译实例巩固鲲鹏Maven仓认识；

目录

Java构建工具

在JAVA开发工具圈中,目前最主流的有以下三个开发工具，依赖管理已经成为了项目构建自动化工具中的一个主要部分。

Ivy 依赖

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!--
https://mvnrepository.com/artifact/com.esotericsoftware.kryo/kryo
-->
<dependency org="com.esotericsoftware.kryo" name="kryo"
rev="2.22"/>
```

Maven 依赖

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!--
https://mvnrepository.com/artifact/com.esotericsoftware.kryo/kryo -->
<dependency>
    <groupId>com.esotericsoftware.kryo</groupId>
    <artifactId>kryo</artifactId>
    <version>2.22</version>
</dependency>
```

Gradle 依赖

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
// https://mvnrepository.com/artifact/com.esotericsoftware.kryo/kryo
compile group: 'com.esotericsoftware.kryo', name: 'kryo', version:
'2.22'
```

Maven介绍

Maven 是 Apache 下的一个纯 Java 开发的开源项目，基于项目对象模型（缩写：POM），
可以对 Java 项目进行构建、依赖管理

Maven官网链接： http://maven.apache.org/

Maven下载链接： http://maven.apache.org/download.cgi

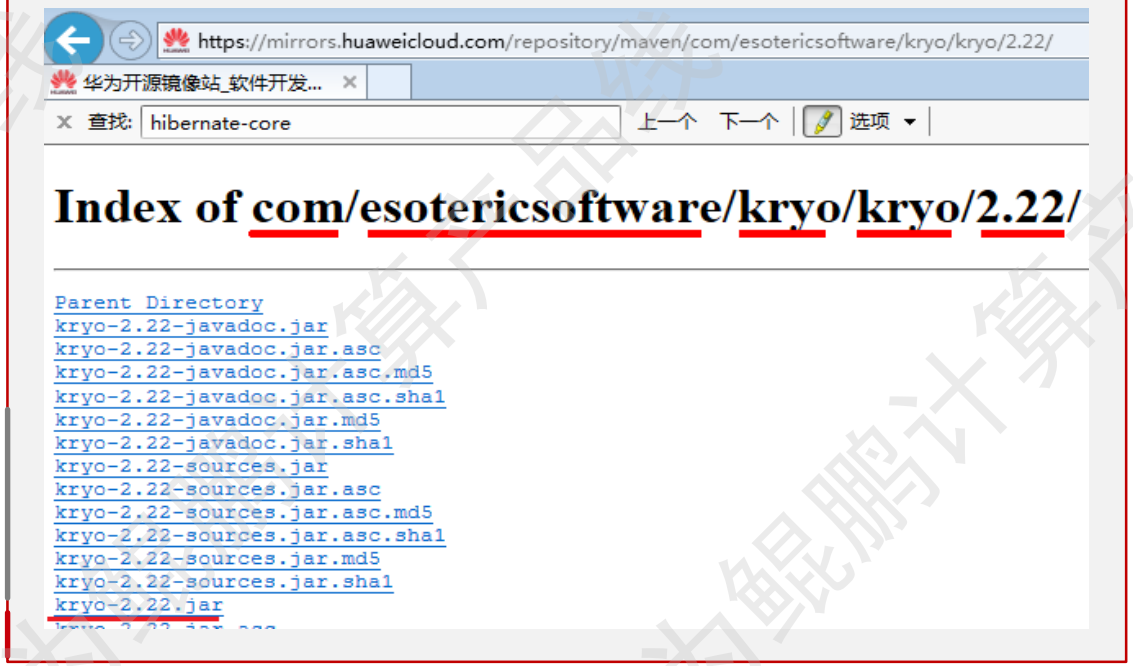
Maven安装指导： http://maven.apache.org/install.html

Maven依赖管理

在Java世界中，可以用groupId、artifactId、version组成的Coordination（坐标）唯一标识一个依赖。

pom.xml文件中一个典型的依赖引用如下图，Maven编译时会自动拼接路径和文件名，去本地或远程仓查找。

```
<!-- Hadoop dependency management is done at the bottom under profiles -->
<dependencyManagement>
  <dependencies>
    <!-- dependencies are always listed in sorted order by groupId, artifectId -->
    <dependency>
      <groupId>com.esotericsoftware.kryo</groupId>
      <artifactId>kryo</artifactId>
      <version>${kryo.version}</version>
    </dependency>
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>${guava.version}</version>
    </dependency>
    <dependency>
      <groupId>com.google.protobuf</groupId>
      <artifactId>protobuf-java</artifactId>
      <version>${protobuf.version}</version>
    </dependency>
    <dependency>
      <groupId>com.google.code.tempus-fugit</groupId>
      <artifactId>tempus-fugit</artifactId>
      <version>${tempus-fugit.version}</version>
    </dependency>
    <dependency>
      <groupId>com.googlecode.javaewah</groupId>
      <artifactId>JavaEWAH</artifactId>
      <version>${javaewah.version}</version>
    </dependency>
```



存储这些组件的仓库有远程仓库和本地仓库之分

Maven仓库分类

Maven仓分为：

本地仓库

存储在本地磁盘
默认在${user.home}/.m2下

远程仓库

一般使用国内镜像或者公司自己搭建私服，可以加快jar包下载速度。

中央仓库

Maven团队维护的jar包仓库
https://repo1.maven.org/maven2/

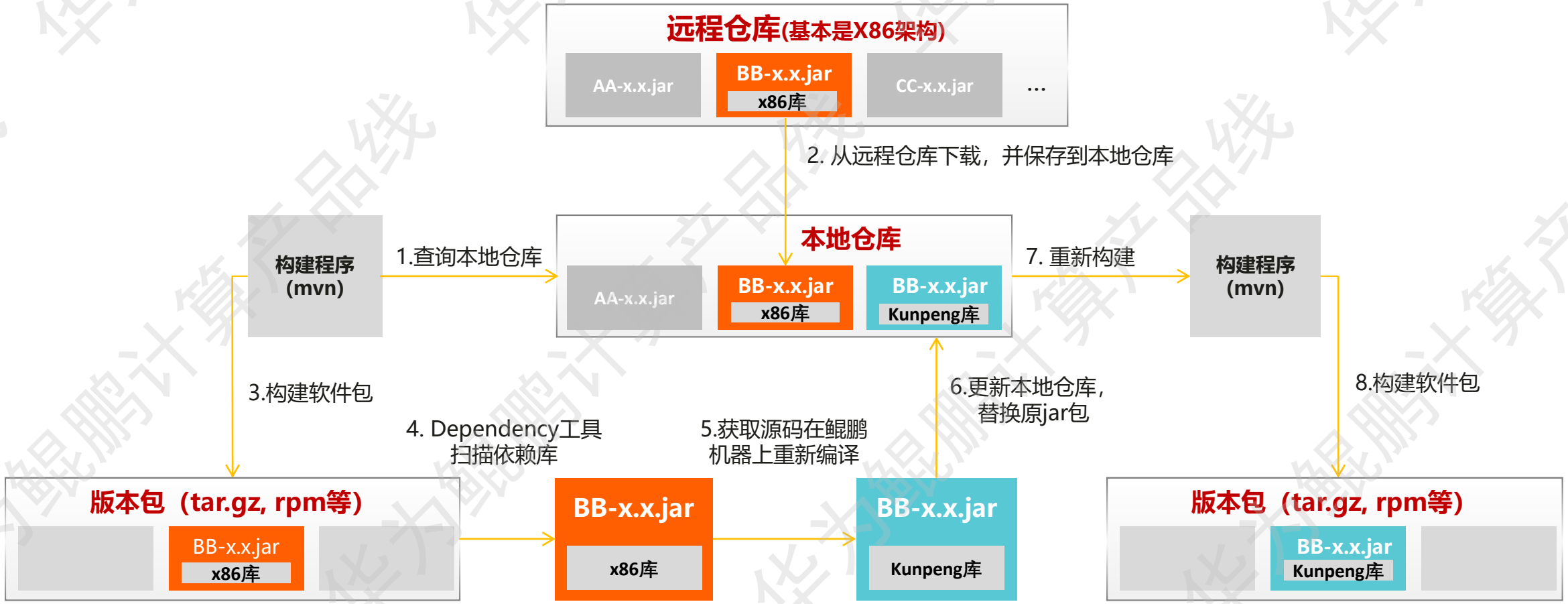
Maven仓库搜索顺序

Maven仓如何搜索：

本地仓库搜索	远程仓库搜索	中央仓库搜索
• 本地仓库找到，直接返回 • 本地仓库没有找到，去远程仓库搜索	• 没有配置远程仓库，去中央仓库搜索 • 远程仓库找到，下载到本地仓库 • 远程仓库没有找到，搜索下一个远程仓，依次类推，如果所有远程仓都未找到，搜索中央仓	• 中央仓库找到，下载到本地仓库 • 中央仓库没有找到，前台打印错误信息

Maven仓库软件构建流程

Maven软件构建关键流程：将X86依赖文件替换成Kunpeng依赖文件，重新构建，直到不包含X86依赖



目录

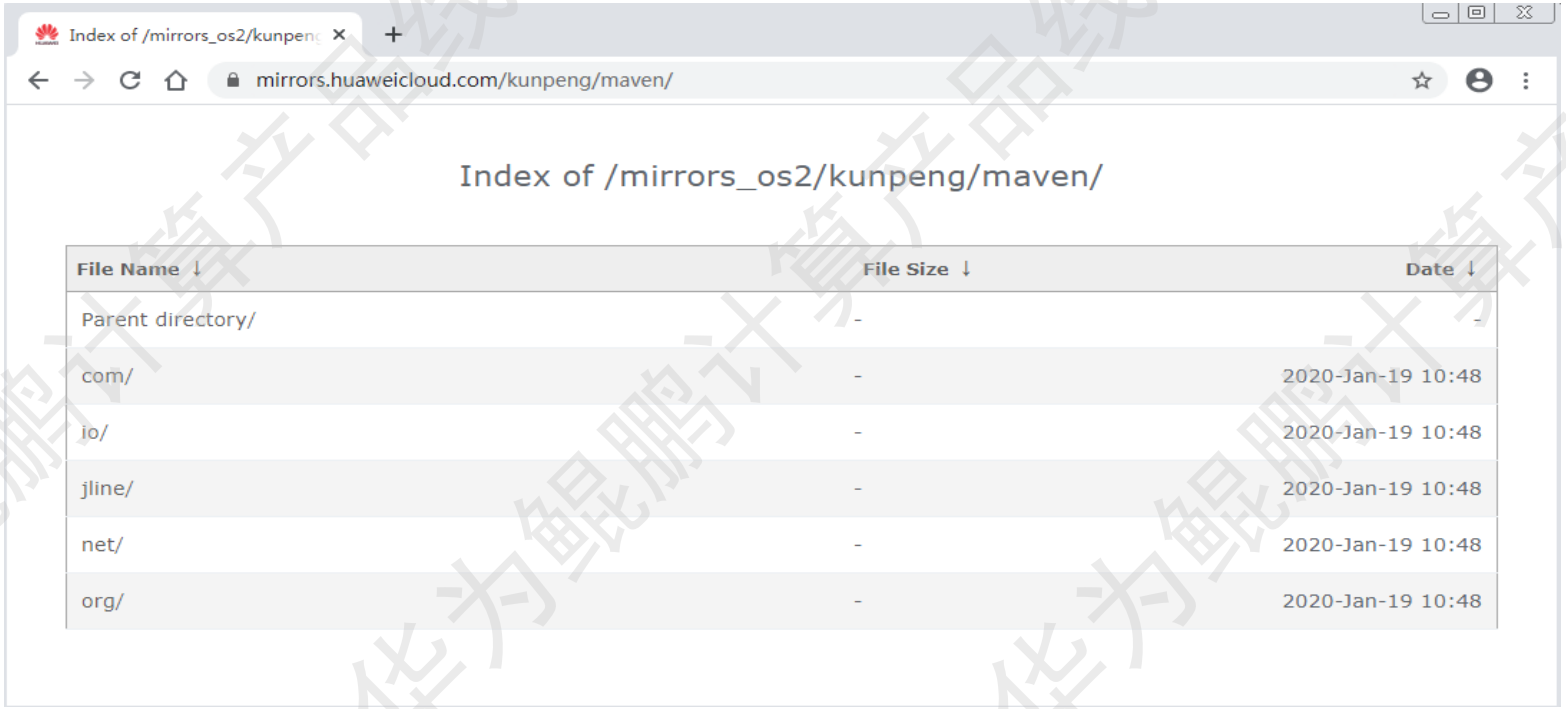
鲲鹏Maven

Maven仓部分jar包依赖x86 so，无法在鲲鹏上直接使用，需要在鲲鹏上重新编译，部分jar包已编译好放在鲲鹏maven仓内，可以直接使用。

鲲鹏Maven链接：https://mirrors.huaweicloud.com/kunpeng/maven/

Kunpeng

远程仓库



目录

如何配置优先搜索鲲鹏Maven仓

前面已了解Maven仓库搜索顺序，可以将鲲鹏Maven远程仓库放在首位，以便Maven优先下载鲲鹏平台jar包。由于鲲鹏Maven仓只放了arm相关jar，所以jar包不全，可以配置第二个Maven远程仓库，当鲲鹏Maven仓搜索不到时，会自动搜索下一个Maven远程仓库。

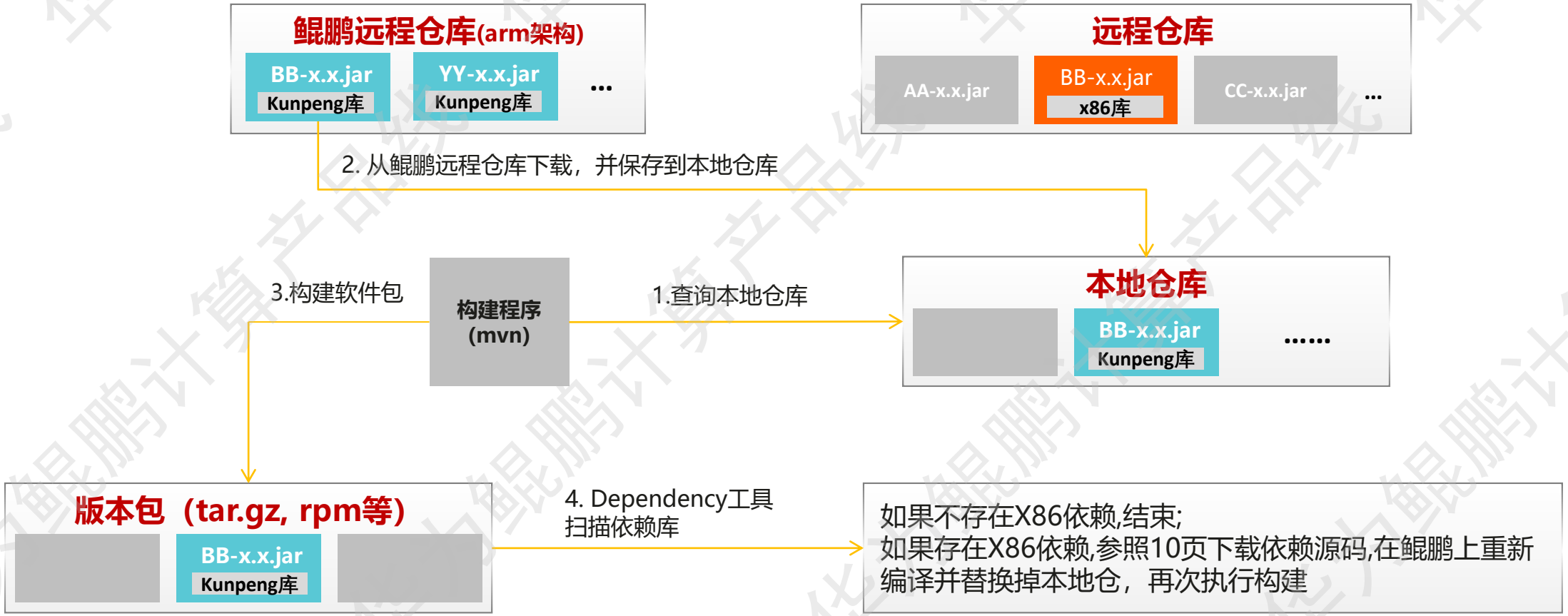
配置方法

- 1. 编辑配置文件${maven.home}/conf/settings.xml
- 2. profiles标签下增加鲲鹏Maven仓

```
<profile>
    …
    <repositories>
     <repository>
      <id>kunpeng</id>
      <url>https://mirrors.huaweicloud.com/kunpeng/maven/</url>
      <releases>
       <enabled>true</enabled>
      </releases>
     </repository>
     <repository>
      <id>huaweicloud</id>
      <url>https://mirrors.huaweicloud.com/repository/maven/</url>
       …
     </repository>
    </repositories>
  </profile>
```

鲲鹏Maven仓库软件构建流程

鲲鹏Maven软件构建关键流程：直接从鲲鹏远程仓下载ARM依赖文件，无需重新编译依赖文件



目录

Hive编译指导

以鲲鹏论坛hive 2.6.3为例（https://bbs.huaweicloud.com/forum/thread-41221-1-1.html），此工程多个jar包含x86架构so，而此部分jar已经过适配并上传到鲲鹏maven仓，编译时只需优先搜索鲲鹏maven仓。

使用如下链接下载HDP x86 RPM包：http://public-repo-1.hortonworks.com/HDP/centos7/2.x/updates/2.6.3.0/HDP-2.6.3.0-centos7-rpm.tar.gz，分析hive_2_6_3_0_235-1.2.1000.2.6.3.0-235.noarch.rpm得出需要移植的第三方依赖包如下表所示

原始jar	so文件
jline-2.12.jar	libjansi.so
snappy-java-1.0.5.jar	libsnappyjava.so
netty-all-4.0.52.Final.jar	libnetty_transport_native_epoll_x86_64.so
leveldbjni-all-1.8.jar	libleveldbjni.so

鲲鹏Maven仓编译Hive

1
安装OpenJDK

2
安装Maven并
配置鲲鹏Maven仓

3
Hive编译

安装OpenJDK和Maven参见（https://bbs.huaweicloud.com/forum/thread-41221-1-1.html）

编译搜索路径如下：

```
[root@centos-164 hive-release-HDP-2.6.3.0-235-tag]#
[root@centos-164 hive-release-HDP-2.6.3.0-235-tag]#
[root@centos-164 hive-release-HDP-2.6.3.0-235-tag]#
[root@centos-164 hive-release-HDP-2.6.3.0-235-tag]# mvn clean install


[INFO] Scanning for projects...
Downloading from kunpeng: https://mirrors.huaweicloud.com/repository/m
^C[root@centos-164 hive-release-HDP-2.6.3.0-235-tag]# mvn clean instal
-Phadoop-2 -DskipTests -Pdist
[INFO] Scanning for projects...
Downloading from kunpeng: https://mirrors.huaweicloud.com/repository/m
aven/kunpeng/org/apache/apache/14/apache-14.pom
Downloading from huaweicloud: https://mirrors.huaweicloud.com/reposito
ry/maven1/org/apache/apache/14/apache-14.pom
Downloading from datanucleus: http://www.datanucleus.org/downloads/mav
en2/org/apache/apache/14/apache-14.pom
Downloading from central: https://repo.maven.apache.org/maven2/org/apa
che/apache/14/apache-14.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apac
he/apache/14/apache-14.pom (15 kB at 11 kB/s)
[WARNING]
[WARNING] Some problems were encountered while building the effective
model for org.apache.hive:hive-metastore:jar:1.2.1000.2.6.3.0-235
[WARNING] 'build.plugins.plugin.(groupId:artifactId)' must be unique b
ut found duplicate declaration of plugin org.apache.maven.plugins:mave
n-jar-plugin @ org.apache.hive:hive-metastore:[unknown-version], /home
/xiongwei/Hive/hive-release-HDP-2.6.3.0-235-tag/metastore/pom.xml, lin
```

备注：Datanucleus为hive pom.xml配置的远程仓库，central为maven默认远程仓。

本章总结

- 本章节简要介绍如何在鲲鹏上使用鲲鹏Maven仓构建Maven工程。

目录

HUAWEI

前言

常见的Linux发行版主要分为两类：类RedHat系列和类Debian系列。类RedHat系统中，软件包的格式是rpm；类Debian系统中，软件包的格式是deb。类RedHat系统提供了rpm（全称是：RedHat Package Manager）命令来安装、卸载和升级rpm软件包；类Debian系统提供了dpkg命令来安装、卸载、升级deb软件包。

分类	发行版	手动安装命令	自动安装命令	软件包后缀
类RedHat	Fedora/CentOS	rpm	yum	*.rpm
	openSUSE/SUSE		zypper	
	Mandriva Linux/Mageia		urpmi	
类Debian	Debian/Ubuntu	dpkg	apt-get	*.deb

本章节主要讲述如何将x86平台软件包rpm迁移（重构）到鲲鹏平台

目标

学完本章节后，您将能够：

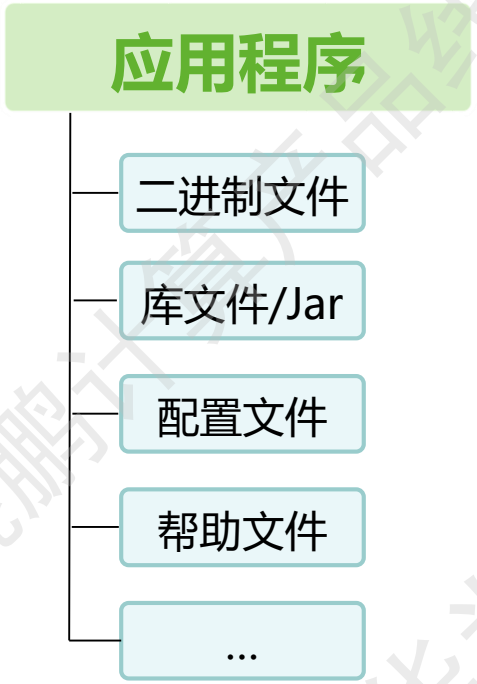
- 熟悉rpm软件包重构流程；
- 使用Porting Advisor开发工具自动将x86 rpm软件包迁移到鲲鹏平台

目录

rpm软件包组成

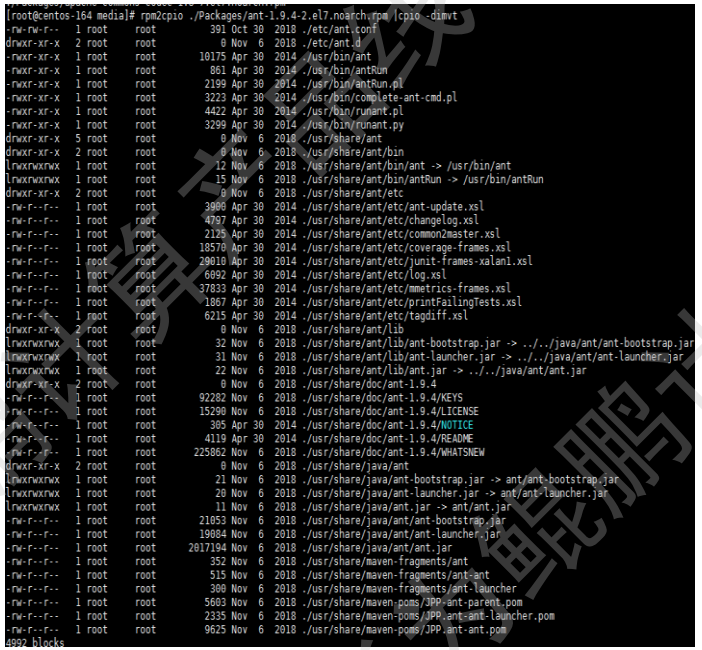
应用程序

常见的开发语言有C、C++，Java、Python等，最终编译成应用程序，应用程序主要包括



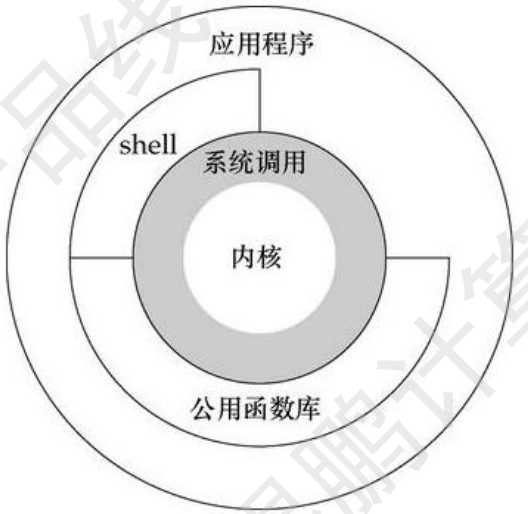
rpm软件包文件组成

rpm可以将应用程序打包，所以rpm包通常包含以上文件(二进制文件,so库文件，Jar包，配置文件等)。



二进制和库文件

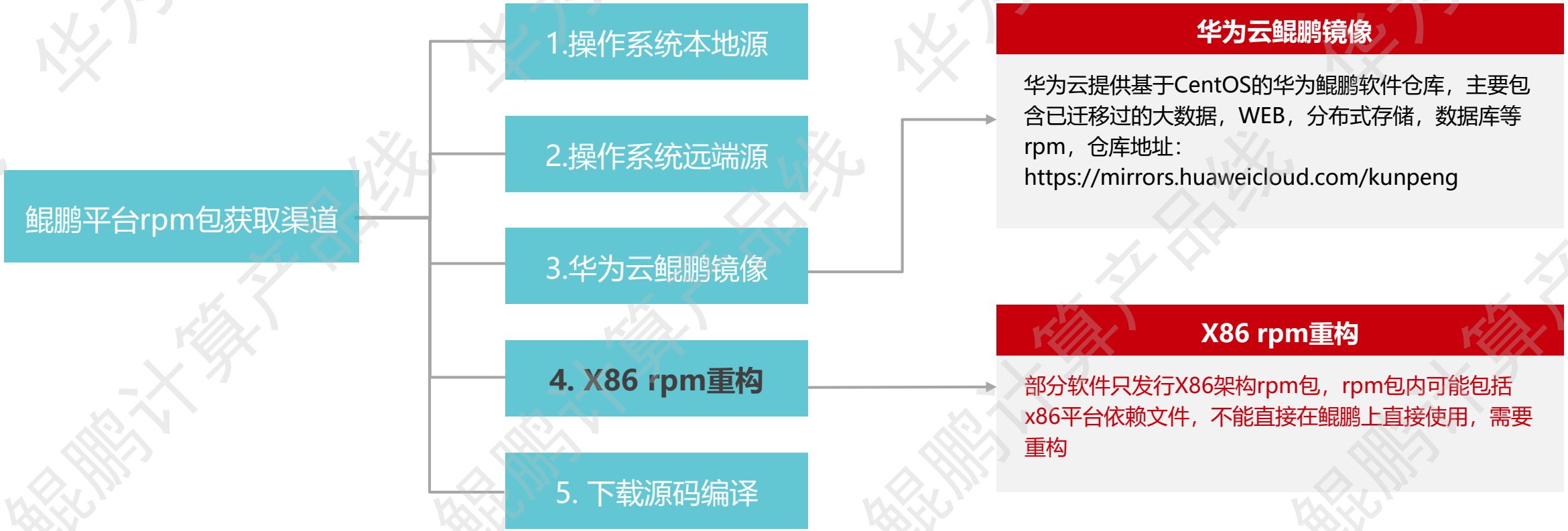
rpm包中与处理器架构相关包括二进制（执行文件），库文件



所以将X86的rpm包重构到Arm的rpm包，需将rpm包含有X86的so、二进制文件，替换成arm架构so、二进制文件

rpm软件包获取渠道

鲲鹏平台的rpm获取渠道



本文主要讲述如何将x86架构rpm重构成鲲鹏平台rpm

目录

传统rpm重构流程

将X86 rpm包重构成鲲鹏rpm包流程：

主要步骤	主要步骤	主要步骤	主要步骤
• 工具扫描x86 rpm识别x86依赖文件	• 如果是jar依赖文件，从鲲鹏Maven仓上查找，或者鲲鹏上重新编译 • 如果是so或其它二进制依赖文件，鲲鹏上重新编译	• 解压 x86 rpm并将x86依赖文件替换成阶段二鲲鹏Maven仓查找到的文件或鲲鹏上重新编译的文件 • 重新打包	• 重新扫描，确认是否还有x86依赖文件 • 安装验证

扫描（扫描X86 RPM，识别x86架构依赖文件）

扫描

下载X86软件包

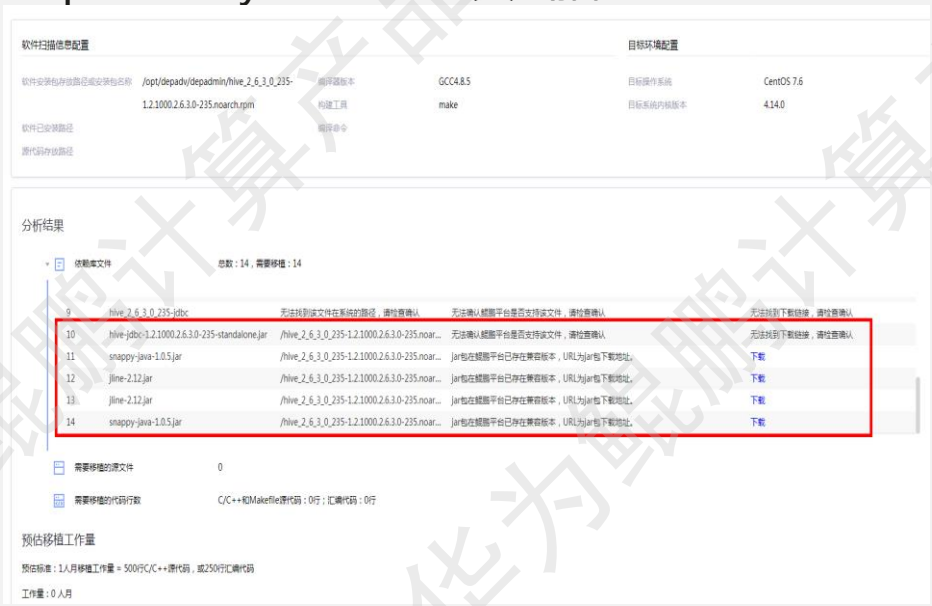
Dependency Advisor工具扫描

步骤说明

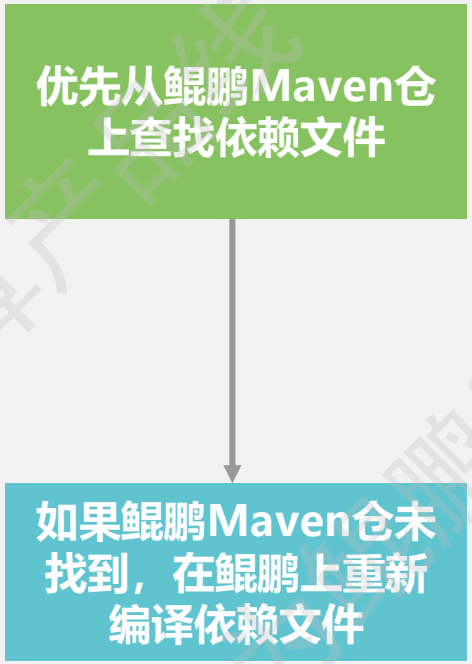
1、下载X86软件包

下载x86 rpm软件包到 /opt/depadv/depadmin目录下（depadmin为登陆用户名）

2、Dependency Advisor工具扫描



编译（鲲鹏上重新编译x86依赖文件）

编译	步骤说明
	1、部分jar包已编译好放在鲲鹏maven仓，可以直接下载，减少重复劳动 查找路径https://repo.huaweicloud.com/repository/maven/kunpeng （鉴于Maven上没有文件列表，查找困难，可以找鲲鹏小智 打开鲲鹏小智：http://ic-openlabs.huawei.com/chat/#/； 输入依赖文件名，根据下载提示下载） 2、对于so，二进制文件或无法从Maven仓上找到的jar文件，需在鲲鹏上重新编译 1）. 打开鲲鹏小智：http://ic-openlabs.huawei.com/chat/#/ 2）. 输入依赖文件名，根据提示搜索移植指南 3）. 如果未找到移植指南，根据源码问题提示编译

打包（鲲鹏上重新生成rpm包）

打包

步骤说明

1、生成spec文件

1）.执行rpmrebuild -s xxx.spec -p xxx.rpm 得到rpm包对应的SPEC文件

2）.修改spec文件中的x86相关字段改为aarch64（noarch、x86_64字样均修改为aarch64），如果spec文件中包含了GLIBC版本信息，需要将GLBC版本修改为GLIBC2.17（CentOS 7.6）

3）. 将spec文件拷贝到/root/rpmbuild/SPECS目录

2、解压x86 rpm

执行rpm2cpio xxx.rpm | cpio -dim解压RPM包，

3、包替换x86 依赖文件

将编译好的组件替换掉RPM包中对应文件

4、打包

1）. 在/root/rpmbuild/BUILDROOT目录新建rpm包名称目录

2）. 将第2步和第3步解压并替换的完整包内容拷贝到，上面目录下

3）. rpmbuild -bb --noclean /root/rpmbuild/SPECS/xxx.spec

验证

验证	步骤说明
	1、重新扫描，确认是否还有x86依赖文件 重复阶段一工作，通过具检查生成的rpm文件是否还包含X86依赖文件，如果还存在，继续后面的编译和打包操作，直到不存在x86依赖文件 2、安装验证 1）. rpm -ivh xxx.rpm，确认是否能安装成功 2）. 运行服务

鲲鹏开发套件Porting Advisor

智能计算开放实验室： http://ic-openlabs.huawei.com/openlab/
鲲鹏开发套件Porting Advisor: https://www.huaweicloud.com/kunpeng/software/portingadvisor.html

Porting Advisor 工具实现迁移自动化

Porting Advisor 工具实现了自动扫描，自动从鲲鹏Maven下载依赖文件，自动打包功能

主要步骤

- ✔ 工具扫描x86 rpm识别x86依赖文件

主要步骤

- ✔ 如果是jar依赖文件，从鲲鹏Maven仓上查找，或者鲲鹏上重新编译
- ✖ 如果是so或其它二进制依赖文件，鲲鹏上重新编译

主要步骤

- ✔ 解压 x86 rpm并将x86依赖文件替换成阶段二鲲鹏Maven仓查找到的文件或鲲鹏上重新编译的文件
- ✔ 重新打包

主要步骤

- 重新扫描，确认是否还有x86依赖文件
- 安装验证

目录

环境准备

环境要求

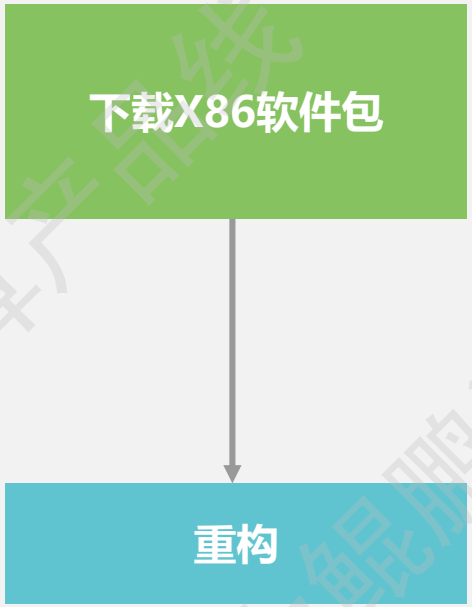
- **TaiShan 200服务器**
- **CPU：Kunpeng 920**
- **操作系统*：**
  - CentOS 7.6
- **远程SSH登录工具已经在本地安装**

安装3个依赖软件

- **安装rpmbuild**
  - #yum install rpmdevtools
  - #rpmdev-setuptree
- **安装rpmrebuild**
  - #mkdir rpmrebuild
  - #cd rpmrebuild
  - # wget https://sourceforge.net/projects/rpmrebuild/files/rpmrebuild/2.14/rpmrebuild-2.14.tar.gz
  - #tar xvfz rpmrebuild-2.14.tar.gz
  - make; make install
- **安装rpm2cpio： CentOS 7.6自带，无需安装**
- **安装Porting Advisor**
  - https://www.huaweicloud.com/kunpeng/software/dependencyadvisor.html下载安装。

Porting Advisor快速重构rpm包流程

X86 rpm在鲲鹏平台重构



步骤说明

1、下载X86软件包

下载x86 rpm软件包到 /opt/portadv/portadmin目录下（portadmin为登陆用户名）

2、重构

1）. 登录Porting Advisor，进入”软件分析构建中心”

2）. 输入下载的软件包名

3）. 点击“构建软件包”

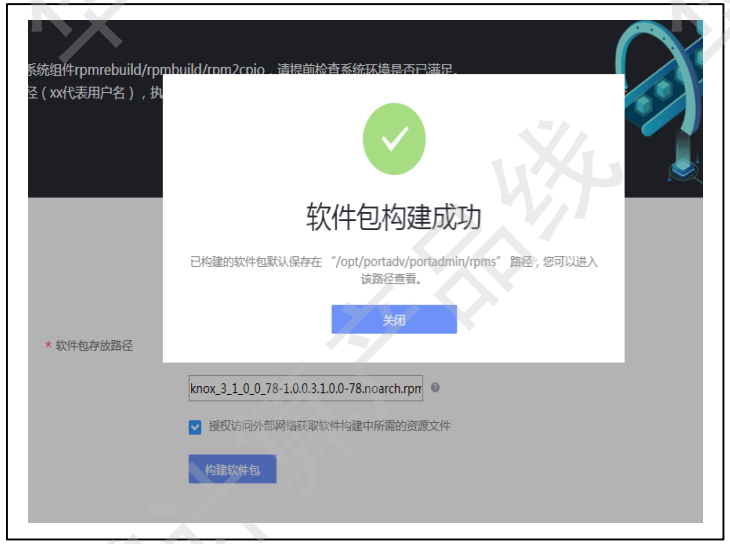
4）. 如果构建成功，工具会弹出成功提示；如果构建失败，根据工具提示执行操作。

软件分析构建中心实例

1 下载X86软件包

```
[root@centos-164 portadmin]#
[root@centos-164 portadmin]# pwd
/opt/portadv/portadmin
[root@centos-164 portadmin]# wget "http://s3.amazonaws.com/dev
.hortonworks.com/HDP/centos7/3.x/BUILDS/3.1.0.0-78/knox/knox_3
_1_0_0_78-1.0.0.3.1.0.0-78.noarch.rpm"
--2020-02-13 11:37:20--  http://s3.amazonaws.com/dev.hortonwor
ks.com/HDP/centos7/3.x/BUILDS/3.1.0.0-78/knox/knox_3_1_0_0_78-
1.0.0.3.1.0.0-78.noarch.rpm
Connecting to 192.168.102.225:8080... connected.
Proxy request sent, awaiting response... No data received.
Retrying.

--2020-02-13 11:37:42--  (try: 2)  http://s3.amazonaws.com/dev
.hortonworks.com/HDP/centos7/3.x/BUILDS/3.1.0.0-78/knox/knox_3
_1_0_0_78-1.0.0.3.1.0.0-78.noarch.rpm
Connecting to 192.168.102.225:8080... connected.
Proxy request sent, awaiting response... 200 OK
Length: 74795696 (71M) [application/x-rpm]
Saving to: 'knox_3_1_0_0_78-1.0.0.3.1.0.0-78.noarch.rpm.1'

100%[====================>] 74,795,696   576KB/s   in 2m 17s

2020-02-13 11:40:00 (533 KB/s) - 'knox_3_1_0_0_78-1.0.0.3.1.0.
0-78.noarch.rpm.1' saved [74795696/74795696]

[root@centos-164 portadmin]#
```

2 重构



1. 下载X86软件包

#cd /opt/portadv/portadmin

#wget "http://s3.amazonaws.com/dev.hortonworks.com/HDP/centos7/3.x/BUILDS/3.1.0.0-78/knox/knox_3_1_0_0_78-1.0.0.3.1.0.0-78.noarch.rpm"

2. 重构

porting-advisor软件分析构建中心输入“knox_3_1_0_0_78-1.0.0.3.1.0.0-78.noarch.rpm”，点击“构建软件包”

本章总结

- 本章主要帮助学员了解如何通过Porting Advisor开发工具迁移(重构)rpm包。

更多信息

相关链接：

鲲鹏社区首页：https://www.huaweicloud.com/kunpeng/

鲲鹏软件栈：https://www.huaweicloud.com/kunpeng/software.html

鲲鹏伙伴计划：https://www.huaweicloud.com/kunpeng/partners.html

鲲鹏产品与云服务：https://www.huaweicloud.com/kunpeng/product.html

鲲鹏解决方案：https://www.huaweicloud.com/kunpeng/solution.html

鲲鹏论坛：https://bbs.huaweicloud.com/forum/forum-923-1.html

鲲鹏计算产业：https://e.huawei.com/cn/kunpeng


谢 谢
www.huawei.com

附录A.1 常见编译脚本、编译选项移植

功能说明	X86代码	鲲鹏代码
为指定类型处理器要生成相应指令	-march=i386	-march=armv8-a
crc32使能编译选项，支持生成 crc32 指令	-mcrc32	-march=armv8-a+crc32
定义编译生成的应用程序为32位， 编译选项不同	-m32	-mabi=ilp32 (gcc低版本不支持，推荐gcc7.3以上)
指定目标处理器名称， 以使生成的代码性能更优	-mtune=intel	Gcc9.1及以上版本可指定 -mtune=tsv110
使用SSE指令生成浮点运算	-mfpmath=sse	无该编译选项、删除
X86下SSE相关编译选项	-msse/-sse2/-sse4/...	删除该部分编译选项
指定生成代码的大小端序	默认是小端序	指定大端序 -mbig-endian 指定小端序 -mlittle-endian
Char数据类型修改	默认char为有符号，无需指定	-fsigned-char(默认char无符号)

附录A.2 常见builtin函数

无需移植的常用builtin函数列表

功能说明	X86/鲲鹏 代码
计算变量x（uint32）的二进制中1的个数	__builtin_popcount（uint32 x)
按字节翻转x(uint32)，返回翻转后的结果	__builtin_bswap32 (uint32_t x)
按字节翻转x(uint16)，返回翻转后的结果	__builtin_bswap16 (uint16_t x)
变量x中1的个数的奇偶性	__builtin_parity(x)
判断type1和type2是否是相同的数据类型	__builtin_types_compatible_p(type1, type2)
引导gcc进行条件分支预测	__builtin_expect (long exp, long c)
返回x中最后一个为1的位是从后向前的第几位	__builtin_ffs(x)
判断exp是否在编译时就可以确定其为常量	__builtin_constant_p (exp)
变量x二进制下末尾0的个数	__builtin_ctz(x)
变量X二进制下前导0的个数	__builtin_clz(x)
内存数据预取到cache中	__builtin_prefetch(x)

附录A.3 常见内联汇编函数移植

功能说明	X86代码	鲲鹏代码
指令给处理器提供提示，以提高spin-wait循环的性能	__asm__ (“pause" : : : memory")	__asm__ ("yield" : : : memory")
rep为x86的重复执行指令，需替换为ARM64的rept指令	#define nop __asm__ __volatile__("rep;nop": : :"memory")	#define __nops(n) ".rept " #n "\nnop\n.endr\n" #define nops(n) asm volatile(__nops(n))
获取精准时间戳	__asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));	鲲鹏小智搜索rdtsc，提供3种替换方案
计算8bit数的crc32校验值	__asm__("crc32b %1, %0" : "+r"(crc) : "rm"(v))	__asm__ crc32cb %w[c], %w[c], %w[v]":[c]"+r"(crc):[v]"r"(v))
计算16bit数的crc32校验值	__asm__("crc32w %1, %0" : "+r"(crc) : "rm"(v))	__asm__ crc32ch %w[c], %w[c], %w[v]":[c]"+r"(crc):[v]"r"(v))
计算32bit数的crc32校验值	__asm__("crc32l %1, %0" : "+r"(crc) : "rm"(v))	__asm__ ("crc32cw %w[c], %w[c], %w[v]":[c]"+r"(crc):[v]"r"(v)
计算64bit数的crc32校验值	__asm__("crc32q %1, %0" : "+r"(result) : "rm"(v))	__asm__ ("crc32cx %w[c], %w[c], %x[v]":[c]"+r"(crc):[v]"r"(v))
对整数变量进行原子加	asm (LOCK_PREFIX "addl %1,%0" : "+m" (v->counter) : "ir" (i))	__sync_add_and_fetch(&((*v).counter), 1)
使用gcc的builtin，把内存数据预取到cache中	asm volatile("prefetcht0 %0" :: "m" (*(unsigned long *)x))	__builtin_prefetch(_x)