1.   综述

应用的迁移主要是系统软件层面的移植（指将某个环境中的事物移动到其他环境，并仍可正常发挥作用）与迁移（指将某个事物的运作方式移动到另一个事物上），而调优则包含国产平台上主机硬件层、操作系统层与软件本身的参数优化。

* 为什么X86应用不能直接运行在ARM平台上？

      例如，使用C语言编写的一个简单的Hello World程序，在X86\_64的机器上编译通过后，将所生成的可执行程序直接拷贝至基于ARM架构芯片的服务器上去执行，会报文件格式不正确之类的错误从而导致程序无法执行，原因是程序的编译平台和和程序的运行平台不同导致的，X86\_64的机器CPU是基于复杂指令集的CISC架构，而基于ARM芯片的服务器的CPU是基于精简指令集的RISC架构的，这两种指令集定义以及执行读取等寻址方式均不同，所以基于其中一种指令集编译得到的程序二进制文件，无法被另一种指令集的机器所理解，也就无法在后一种机器上运行相应的应用。

2.   操作系统软件迁移与移植

2.1.  软件移植到国产操作系统的迁移过程

2.1.1.  软件迁移过程概述

对用户而言，在进行国产化的过程中，底层硬件与操作系统的变更，会涉及上层应用的全面适配与兼容需求，因此进行兼容性评估和适配工作，需要经过以下几个步骤：

1.     软件移植系统信息收集

信息收集的目的是进行难点分析及工作量评估，而需要收集的信息主要包括如下内容：

Ø  硬件信息：整机型号（服务器/终端）、外设、配件等信息；

Ø  软件信息：虚拟化/云平台、操作系统、中间件、数据库、浏览器、控件、第三方软件（开源/商业/自研）、应用架构（B/S、C/S）等信息；

Ø  运行环境：开发语言、开发框架、底层组件、依赖库等信息。

2.     软件迁移分析

在移植前的信息收集完成之后，需要对这些信息进行分析，分析内容主要包括：软件技术栈、开发框架、开发语言、依赖基础组件、依赖库、以及软件代码。

3.     迁移支撑

在软件迁移过程中，操作系统厂商可提供迁移支撑服务，包括：

Ø  组件编译：提供应用底层所需组件支撑；

Ø  技术咨询：提供系统支撑组件包的技术咨询；

Ø  问题处理及协助：针对能定位到具体模块的问题，可提供Demo进行复现分析

Ø  开发文档：提供系统相关API文档和技术文档的输出

4.     性能调优

软件迁移完成，除满足功能需求外，还要在目标系统中满足性能指标要求，通常的性能调优过程如下：

Ø  性能指标测试

Ø  性能指标单项优化

Ø  软件系统业务综合指标优化

5.     测试与认证

软件迁移完成，在软件规模上线应用前，除功能测试、性能测试，同时还要进行压力测试、稳定性测试，保证业务压力测试指标要求。

2.1.2.  迁移前的分析与评估

1.    技术框架

请首先确认产品在原系统上使用的技术框架，在目标系统上是否存在。

Ø  跨平台技术框架

如果采用了跨平台技术框架，比如QT，那么移植相对容易一些。

但需要注意，如果是C/C++技术框架，其SDK是否提供支持目标系统的预编译二进制文件，如果没有，需要首先将技术框架必要的源码进行编译。

如果是纯java或lua之类的解释型语言框架，则无此必要。

如果是混合技术框架，比如java框架中调用了C的类库，需要特别注意类库问题，具体请查看下一小节。

Ø  非跨平台技术框架

如果您的应用使用了WPF这一类特定平台深度绑定的技术框架，而这种技术框架并不支持目标系统，那就需要考虑进行替代开发。

替代开发可以分开来看：

图形库：图形库无法直接迁移，需要重新替代开发，推荐使用qt或electron。

逻辑库：如果使用通用C/C++逻辑库，比如gnu stdc++，可以尝试直接对逻辑库进行绑定开发。

2.    开发辅助工具

需要确认在Windows上的开发辅助工具，在目标系统是否存在，或有替代工具。

2.1.3.  分析软件包依赖库

Linux和Windows分析依赖库的方法是一致的，具体分两点：

1.     开发依赖

如果是C/C++源码，依赖库重点查看头文件，需要查看需求的头文件是否齐备以及版本是否符合。首先查询头文件所属软件包、以及软件包版本，确认版本符合后，安装软件包。

2.     部署依赖

部署时查询方式一致，只是查询的是.so库文件所属包，并填入control文件的depends字段内。

2.1.4.  分析待迁移软件的源码文件

如果是C/C++源码，首先确认是否使用平台特有API，比如win32 API，需要替换为gnu libc/libc++兼容的跨平台api库，例如Direct2D可以考虑替换为SDL。

其次注意是否使用了汇编代码，由于主流国产操作系统支持多架构，尽量避免使用汇编，建议最低级的语言为C。

分析源码文件，如果是C++源码部分，可以考虑按照类逐个分析，提取核心类，优先尝试进行迁移。

一款软件中，一般可以分类为图形渲染引擎、数据库、逻辑脚本库、网络协议库、文字读取和渲染，登录界面等。

Ø  图形渲染引擎：Windows上最知名和常用的directX，主流国产操作系统并不支持，考虑替换为OpenGL或vulkan。

Ø  数据库：一般用作本地数据存储和校验，可以考虑替换为sqlite。

Ø  逻辑脚本库：可以考虑替换为LUA相关的技术框架。

Ø  网络协议库：分为通信库和加密库，加密库一般用的是openssl，这是跨平台的；主流国产操作系统有很多通信库，TCP、UDP支持的都有。

Ø  登录界面：Windows上用wpf的比较多，可以考虑替换为qt或electron。

Ø  字体读取和渲染：可以考虑用freetype替换。

特别注意：如果实在不知道C/C++代码如何迁移，请全部代码用标准C/C++语法和跨平台的类库进行编写。推荐使用C/C++ 11标准的规范，不要使用C/C++ 17或更新。

2.1.5.  平台迁移代码量和工作量

建议按照功能模块，通过分类，进行评估。

每个模块的迁移量和工作量，依据迁移模块数量、开发人员数量、学习时间等维度综合评估。如果有跨平台可重复使用的模块，迁移量会减少。

一般情况下，如果模块难以迁移，工作量可以按照该模块重写一遍的历史工时进行参考计算。

2.1.6.  迁移步骤及具体方法

★ 首先按照功能，对产品进行模块分类。

★ 其次，每一个模块，判断其使用的技术是否跨平台，不是跨平台的部分，挑选替代品继续进行评估。

★ 再次，优先迁移核心模块，确保软件最基础部分能够成功迁移，然后可以将其他模块逐渐迁移完善。

★ 优先在AMD64架构下进行迁移，后续可以针对其他CPU架构编译后，再进行测试和调整。

★ 最后，测试稳定性和性能，然后进行打包。

2.2.  WINDOWS应用分析和移植方式

Windows的系统开发大约已经有二十多年的时间，已经积累了大量的应用开发工具，以及在此基础上开发的应用程序。Windows的应用开发技术也经历了很多变迁，从单机程序发展到C/S架构、B/S架构等。下面就这些应用使用到的语言和技术来分析对应的移植方法。

1.     单机版应用

单机版应用程序相对比较简单，一些老的MIS系统等均为单机版应用。

对于仍在使用的DOS应用程序，在国产操作系统下可以使用dosbox等虚拟环境来运行。

2.     C/S架构的Client端

Client端编程使用的语言和技术都很广泛，针对应用程序所使用的语言不同，移植方法也不同。如果应用程序是由C++编写并且带有图形界面，可以考虑使用跨平台的开发环境QtCreator、Eclipse等，其中根据功能调用或者使用的技术对应主流国产操作系统下的解决方法如下：

Ø  网络接口：支持Berkeley套接字编程，提供C编程接口；提供EPOLL传输框架，同样具备高并发处理能力。

Ø  图形接口：可以替换为gtk，qt，wxWidgets，JAVA等。

Ø  音频接口：可以通过内核提供的系统调用访问声卡驱动，也可通过编程接口如alsa、sox等实现。

Ø  数据库：在国产操作系统上使用国产达梦、金仓或者开源的mysql等，接口有unixODBC，SQL等。

Ø  并发多任务：在主流国产操作系统上同样支持进程/线程概念，符合POSIX标准，支持IPC机制。

Ø  注册表：在主流国产操作系统上每个用户可以使用gconf机制，共用的环境变量可以根据系统配置文件管理。

Ø  MFC，ATL，WTL：可以使用跨平台开发环境，部分需要根据功能编写Linux的插件或corba组件。其中MFC应用可以相对容易地迁移为wxWidgets。

Ø  DirectX：可以使用第三方软件SDL、OpenGL等。

Ø  COM、ActiveX：根据功能不同可以对应到corba、widget、插件等。

有些开发语言的某些版本如果有Linux下相应的开发工具可以对其代码进行重新编译。比如：

Ø  Delphi/C++ builder：kylix。

Borland发布到kylix 3之后没有推进新的版本，所以kylix的使用只能针对Delphi 6及之前的版本。

Ø  VB：可以使用.net迁移向导VB.net，或者使用Gambas（与VB较为类似）重新编写。

在VB编写中，如果没有使用ActiveX、没有和数据库的连接，可移植程度较高。

Ø  Powerbuilder：Sybase曾经推出过Linux版本，并不能保证在相应的国产操作系统版本上能安装，没有对应版本的只能重写代码。

Ø  .net：利用MONO技术迁移到主流国产操作系统平台。

总体来说，这类程序的可移植性较低。

3.     B/S架构的Browser端

主流国产操作系统支持国产360、奇安信浏览器，也支持开源Firefox浏览器，使用中可能遇到以下的问题：

Ø  HTML页面：部分程序页面不符合W3C标准，需要进行测试和少量迁移。

Ø  Java Applet：一般浏览器需要安装相对应的支持库。

Ø  页面内嵌多媒体：主流国产操作系统支持多种媒体格式，如果网页没有调用具体的控件，则网页不需要修改，否则就需要代码调整。

Ø  HTC组件：在主流国产操作系统下可利用moz-behavior特性进行迁移。

Ø  ActiveX控件：只能重新开发为对应的浏览器插件。

2.3.  应用移植分类小结

从上面的分析可以知道，不同的Windows应用移植方法，可移植的程度都不同。从移植的程度来说，分为以下几类：

1.    完全移植

在主流国产操作系统下均以本地代码运行，应用程序安全可靠并且运行效率最高。主要包括：

Ø  跨平台的脚本语言（如PHP、Python等）编写的应用程序，几乎无需修改源代码即可在目标系统下运行。

Ø  跨平台的编译型语言（如Java、C++等）编写的应用程序，几乎无需修改源代码，在目标系统下重新编译即可运行。

Ø  VC、C#、.net等非跨平台开发语言编译的应用程序，在目标系统下需重新开发，成本较高。

2.    虚拟化

使用虚拟化技术是在国产操作系统上再运行一个Windows，无需移植，保证完美运行。虽然虚拟机是效率损失最大的，但是随着硬件虚拟化技术的出现使得效率损失减少，尤其在多核CPU出现后，这种效率损失几乎忽略不计。

不常使用的应用程序，可以应用这种方式，例如某些带ActiveX控件的网站。

在实际移植过程中，要考虑性能、稳定性和移植代价之间的平衡。根据应用、使用环境不同，选择相应的移植方式。

2.4.  应用移植技术难点

1.     硬件适配

硬件适配主要涉及板卡移植和外设移植，由于硬件设计细节由硬件开发商掌握，操作系统厂商仅能根据公开的技术资料开发驱动。一直以来，Windows操作系统都占据市场主导地位，硬件厂商通常自主为其提供驱动，所以在Windows上的硬件能力发挥稳定；而国产操作系统平台暂时不具备市场号召力，只能根据部分硬件资料，参考公开可获得的驱动程序源代码，进行驱动开发和移植工作，通常新驱动都无法完全发挥硬件的全部能力。如果没有相关硬件资料和驱动源代码，通常必须重新选型。

2.     运行支撑库

无论是Windows还是国产操作系统都提供一套运行库，提供应用软件调用，其中尤以3D图形库和多媒体库的需求比较突出。主流国产操作系统提供了一套与Linux兼容的基于XWindow系统的标准图形库，与Unix平台的XWindow接口基本一致。主流国产操作系统所提供的3D图形库主要是来自开源社区的Mesa3D，它是目前唯一免费和开源的OpenGL实现。使用广泛的Windows平台下Direct3D引擎目前暂无对应的适配国产软硬件平台的兼容版本。不过，可以利用开源社区Gallium3D项目。

多媒体库主要提供多媒体格式的编解码器，由于公开资料中相关资源较为丰富，国产操作系统对此的支持较好，但对于原先基于国外操作系统开发的软件，其调用接口将需要调整。

其他的运行库还包括线程库、数学库、字符串处理库等，POSIX都对其进行了较为详细的规范。

3.     字体字库

中文应用的字体字库问题通常会给应用迁移过程带来麻烦，因为主流国产操作系统所能支持的字体及其缺省采用的编码和字体，与国外平台均有所不同。

由于版权关系，Windows平台上的“宋体”、“楷体”、“黑体”、“仿宋”等常用字体文件只能采用字形相似的其他字体文件代替，目前因主流国产操作系统提供了方正和华文字库，基本实现了宋楷黑仿主要字体的显示。为解决字体名不同的问题，主流国产操作系统进行了字体映射配置，应用软件沿用Windows平台的字体名时，也能在主流国产操作系统上以相同字体显示。

但部分Windows平台上常用的字体，则没有相应的可获得授权的可选字体，如微软雅黑字体等，即使采用映射的方式选择其他字体，在字形上也会有所区别，对应用软件而言，就有可能在对齐、排版上带来一些问题。

Windows系统缺省编码通常是GB18030（中文系统），主流国产操作系统缺省编码基本统一为UTF8，在与中文编码相关的应用中，如果未采用标准的国际化处理规程，则可能在应用迁移中带来问题。主流国产操作系统提供了一整套国际化和本地化编程接口，应用迁移时应遵循该接口对字符串进行处理，以避免引入编码错乱问题。

2.5.  迁移中遇到的常见问题及解决

1.     什么是依赖，怎么安装依赖？

所谓的依赖，指的是操作系统在线仓库内的软件包，大部分是动态链接库和头文件，可以避免自行编译，节省开发和部署成本。

2.     怎么调试？

调试工具有很多，比如strace和gdb，strace详细记录二进制文件的运行过程，gdb可以排查二进制文件的运行断点。

建议开发过程中编译带debug信息的二进制文件，方便调试。

3.     开发中需要的xx软件没有，怎么办？

如果是软件的组成部分，仓库中并没有的情况，优先考虑自行编译，也可以联系软件作者提供协助；主流国产操作系统厂商可以分派工程师进行在线协助。

3.   服务器上应用软件移植与国产化迁移

3.1.  应用迁移基本过程

一个应用从X86平台到其他平台典型的迁移过程包括如下三个步骤：

1、技术分析阶段：准备调试编译环境，软件栈分析。

2、编译迁移阶段：搭建编译调试环境，重写汇编代码，修改编译选项，代码编译。

3、功能验证：搭建功能测试环境，全量功能验证。

按照程序源代码到机器指令之间翻译方式的不同，高级语言通常可以分为两类：一类是编译翻译，一类是解释翻译，分别对应着编译型语言和解释型语言。

3.1.1.  编译型语言应用迁移

典型的如C、C++语言，都属于编译型语言，源代码到执行的过程概括如图6-2所示。C/C++编译好的程序是机器指令，由操作系统加载到存储器（一般为内存）后由CPU直接执行。

基于编译型语言开发的应用程序，例如C/C++语言应用程序，其编译后得到可执行程序，可执行程序执行时依赖的指令是CPU架构相关的。因此，基于X86架构编译的C/C+ +语言应用程序，从X86平台迁移到国产平台时，必须使用源代码、经过重新翻译才能运行。

C/C++移植流程过程如下。

1.    获取应用程序源代码。

2.    如果源代码涉及到X86汇编的代码需要使用国产架构汇编重写；汇编程序和C/C++源代码中内联汇编需要重写。原因国产架构指令集不兼容，跨平台执行汇编指令会触发指令异常。

3.    根据使用国产平台编译环境，使用相应的编译工具，编译应用程序。 在国产架构编译环境上，编译方法与X86服务器一致，可使用make、cmake、autoconfig等工具进行编译。对于多数开源软件，执行./configure; make; make install。

4.    如遇到开源项目库不支持国产架构出现编译失败（这种情况极少，一般发生在较旧的项目代码），解决方法包括：修改代码，寻求替代库等。

5.    编译时提示代码错误，可能需要根据平台差异修改部分代码，例如平台类型相关宏定义。

6.    应用程序部署：安装部署、设置系统启动脚本等操作方法与X86服务器一致。

3.1.2.  解释型语言应用迁移

基于解释型语言开发的应用程序，与CPU架构/指令集相关的差异由语言运行环境（即虚拟机或者解释器）进行屏蔽/解决，例如Java、Python、PHP等。Java/Python编译好的程序是平台无关的字节码，由虚拟机解释执行，虚拟机完成平台差异的屏蔽。

将这类应用程序移植到国产架构平台，通常需要两步：



* 大多数解释型语言开发的应用程序，无需修改即可在国产架构平台上运行。
* Java应用程序jar包内，可能包含基于C/C++语言开发的so库文件，这类so库需要移植编译，移植so库文件后，重新打包jar包，移植编译so库遇到的问题可以参考编译型语言应用迁移的方法解决。

1.    安装语言执行环境。

对于Java是Java虚拟机，对于PHP是PHP语言解释器，对于其他解释型语言，例如 Python、Ruby、Go等，请安装各语言解释器。各语言虚拟机或解释器，需要国产架构版本，其中大多已被操作系统预置，请使用操作系统包管理器检查。如果系统未预置或者预置版本不符合要求，请使用源代码或包管理器进行安装，并参考本文档相关节内容进行移植。

以UOS使用包管理器安装JAVA虚拟机为例：

yum install java-1.8.0-openjdk

2.    执行应用程序。

将应用程序部署在国产架构服务器上，无需修改和重新编译，按照与X86一致的方式部署和运行应用程序。

3.1.3.  制定整体迁移策略

建议在做应用迁移前，先制定好整体迁移策略：

* 收集软件栈信息。

          作为迁移计划和迁移策略制定的输入，软件栈信息包括当前使用的硬件信息，如芯片，服务器型号。软件信息包含，如OS，数据库、中间件、编译软件，自研软件，开源软件，商用软件等。

* 制定具体的迁移策略和计划。

           根据收集好的软件栈情况，针对不同类型的软件，制定具体的迁移策略和实施计划。

3.2.  应用迁移具体方法

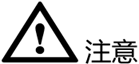
3.2.1.  编译工具

3.2.1.1.      开源编译工具

rpm及deb格式的Linux应用软件二进制安装包，是与操作系统版本及CPU架构相关的，因此为X86服务器编译生成的二进制安装包，是无法在国产架构服务器上使用的，需要使用源代码，通过编译工具软件，重新编译为国产架构服务器版本的安装包。对于不同编译型语言开发的软件，应选择对应的编译器进行二进制编译。

编译器选择如下：

* 基于GO开发的软件，推荐使用Golang编译器1.9以上版本（1.8版本在部分场景下存在性能问题）。
* 基于Fortran开发的软件，推荐使用gFortran编译器。
* 基于C/C++开发的软件，推荐使用GCC。



* 由于不同版本的编译器对语言标准的支持存在差异，故需检查软件源码所使用的语言标准，选择支持该语言标准的版本。

3.2.2.  编译类型

3.2.2.1.      本地编译

本地编译本地运行，即在直接在国产架构环境上利用gcc等工具，编译出来的可运行的程序，编译完成后直接测试运行。

1.     ./configure 执行编译配置。

configure是一个shell脚本，用来检测你的安装平台的目标特征，它可以自动设定源程序以符合各种不同平台上Linux/Unix系统的特性，并且根据系统叁数及环境产生合适的Makefile编译配置文件或是C的头文件(header file)，让源程序可以正确地在这些不同的平台上被编译连接。

2.     make 执行编译

它从Makefile中读取指令，然后编译。

3.     make install执行安装

它也从Makefile中读取指令，安装到指定的位置。

3.2.3.  C/C++代码移植参考

3.2.3.1.      -m64 编译选项

-m64是X86 64位应用编译选项，m64选项设置int为32 bits及long指针为64 bits，为 AMD的X86 64架构生成代码，在国产架构平台无法支持，编译时报错。

解决方法：

将ARM64平台对应的编译选项设置为“-mabi=lp64”

将LoongArch64平台对应的编译选项设置为“-mabi=lp64d”

3.2.3.2.      configure配置与makefile构建

检查软件源代码是否包含Makefile文件。当软件源代码不包含Makefile文件时，用户可利用自动编译工具，例如Autoconf、Automake和Build工具构建Makefile文件。解压源代码后，进入代码目录检查代码结构，查看是否存在configure脚本，再执行 configure命令：

./configure

如果不存在configure脚本，可通过源码中的自动化脚本生成configure文件。

如果执行过程中存在以下问题，对应的解决方案如下：

* configure文件不包括国产架构选项。

           修改configure文件，增加适配国产架构编译选项。

* 无法识别编译类型。

           执行configure脚本时，提示无法识别编译类型，如下所示：

           configure: error: cannot guess build type: you must specify one

           configure: error: ./configure failed for libltdl

           需要在config.guess和config.sub里增加国产架构的支持。

3.2.3.3.      -march与-mtune编译参数不兼容

Makefile中包含“-march”与-mtune编译参数，国产架构平台上编译提示参数不兼容。

解决方法：

兼容的调整后参数为：

ARM平台：

-march=armv8.1-a -tune=cortex-a72

参数设置指导请参考：<https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html>

Loongarch平台：

-march=loongarch64-tune=la464

参数设置指导请参考：[https://gcc.gnu.org/onlinedocs/gcc/LoongArch-Options.html](https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html)

3.2.3.4.      代码中汇编指令需要重写

在追求更高性能或者调用CPU架构相关的特殊指令时，C/C++代码中可以\_\_asm\_\_或asm关键字修饰，直接以汇编指令代码方式实现某些运算功能，这样的汇编代码我们称为“内联汇编”。

由于指令集的差异，X86和国产架构的汇编完全不一样，所以只要涉及X86内联汇编的地方都要移植修改，针对国产架构平台进行相应修改或重写。

内联汇编的写法有两种形式，“asm”和“\_\_asm\_\_”，其中asm关键字是GNU扩展，如果编译时使用-ansi或者-std选项，需要使用\_\_asm\_\_。 volatile和\_\_volatile\_\_是防止编译器优化，通常我们都添加了volatile描述。

asm [volatile] ( AssemblerTemplate

: OutputOperands

[ : InputOperands

[ : Clobbers ] ])

       汇编/指令可参考资料：

1)      X86指令集手册：<https://www.intel.cn/content/www/cn/zh/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2a-manual.html>

2)      ARM指令集手册：[https://developer.ARM.com/docs/100076/0100](https://developer.arm.com/docs/100076/0100)

3)      LoongArch指令集手册：<https://www.loongson.cn/uploads/images/2023102309132647981.%E9%BE%99%E8%8A%AF%E6%9E%B6%E6%9E%84%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C%E5%8D%B7%E4%B8%80_r1p10.pdf>

4)      内联汇编规则：[https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html#Using-Assembly-Language-with-C](https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html)

3.2.4.  Java软件迁移参考

3.2.4.1.      Java开源软件Maven仓库软件迁移

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。通过Maven工具和项目配置能自动化完成对依赖组件的下载和参与构建过程，其中Maven仓的正确配置和使用是构建迁移关键因素。

**Maven仓库搜索顺序**

1)    本地仓库搜索

* 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

2)    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

3)    中央仓库搜索

* 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软件构建迁移关键流程：将X86依赖文件替换成国产架构依赖文件，重新构建，直到不包含X86依赖。

3.2.5.  Python代码移植参考

**移植基本步骤：**

***步骤 1***准备Python运行环境：使用国产架构版操作系统自带的Python。

***步骤 2***设置环境变量：设置Python运行时的环境变量

***步骤 3***【可选】编译生成pyc文件（字节码）

***步骤 4***【可选】移植外部依赖库

***步骤 5***【可选】更新代码中新的外部依赖库调用代码

***步骤 6***测试：执行Python源码程序

3.2.6.  Go代码移植参考

3.2.6.1.      Go编译系统迁移

Go编译系统迁移，首先下载最新的国产架构 golang。

下载地址为：https://golang.org/dl/，国内下载地址：<https://studygolang.com/dl>。

将下载的二进制包解压至安装目录，这里以/usr/local为例，

通常需要设置3个环境变量GOROOT（软件包安装在系统上的位置），GOPATH（工作目录的位置、需要根据实际代码目录设置）和PATH（设置Go的bin目录），在.profile或/etc/profile文件设置环境变量如下：

echo "export GOROOT=/usr/local/go" >> ~/.profile

echo "export PATH=$GOPATH/bin:$GOROOT/bin:$PATH" >> ~/.profile

source ~/.profile

3.2.6.2.      Go汇编迁移

一般情况下，在国产架构平台上搭建好编译系统就能直接编译运行代码，但对于存在Go汇编的情况，即\*.s汇编文件，需额外进行迁移。

现象描述：

源码包含X86 Go汇编时，编译报错：missing function body。

Go中寄存器包括通用寄存器和伪寄存器，以下重点介绍伪寄存器。

* 伪寄存器

           X86平台下Go存在四种伪寄存器FP、PC、SB、SP，实际是对内存位置的一个引用。

           FP（frame pointer）：帧指针，保存参数和本地变量，用来标识函数参数、返回值。

           SP（stack pointer）：栈指针，存放栈的偏移地址，它指向本地栈帧的顶部。

           SB（static base）：静态基指针，用来表示全局的变量或者函数。

           PC（Program counter）：程序指针，即处理器中常见的PC寄存器，存储需执行指令地址。

* 汇编代码迁移

           Go编译器会输出一种抽象的汇编代码，这种汇编并不对应某种真实的硬件架构。Go汇编器使用这种伪汇编，为目标硬件生成具体的二进制文件。伪汇编的优点在于更易将Go移植到新的架构上，但即使是伪汇编，X86平台与国产架构平台还是存在一些差异。

3.2.7.  其它代码移植

3.2.7.1.      开源类软件

其他小语种代码移植：

编译型：go

解释型：lua、ruby、php、perl、typescript、javascript

编译型：源码需要重新编译

解释型：源码不用编译，安装解释器即可

**解释器搭建，程序运行环境准备方法:**

1.    操作系统自带解释器软件，直接安装即可使用；

2.    从官网上下载支持A国产架构的解释器软件包，直接解压安装使用；

3.    从官网下载解释器源码，编译后使用。

3.2.7.2.      闭源类软件

1、 获取国产架构版本：

联系商用软件提供商，获取与国产架构兼容的软件版本，重新部署使用，迁移或置换商用license。

2、 无ARM或LoongArch版本：

原商用软件提供商无法提供国产架构版本的，可以替换为其他支持国产架构软件的替代方案。

©统信软件技术有限公司。访问者可将本网站提供的内容或服务用于个人学习、研究或欣赏，以及其他非商业性或非盈利性用途，但同时应遵守著作权法及其他相关法律的规定，不得侵犯本网站及相关权利人的合法权利。除此以外，将本网站任何内容或服务进行转载，须备注：**该文档出自【faq.uniontech.com】统信软件知识分享平台**。否则统信软件将追究相关版权责任。