关卡 2

基于鲲鹏代码迁移工具的源码迁移

华为技术有限公司

目录

1 实验环境介绍

1.1 实验介绍

1.1.1 实验简介

应用程序迁移时可根据应用的特点来决定如何移植，应用程序是用编程语言编写的，所以首先可以从应用所使用的语言特性分析应用是否合适迁移到鲲鹏平台。如何使用鲲鹏代码迁移工具去完成软件的迁移，是本关所需要掌握的知识。

1.1.2 实验目的

本实验的主要目的是使用鲲鹏代码迁移工具进行即时交流平台后端程序的迁移，实现后台程序在鲲鹏服务器上的运行。

2 鲲鹏代码迁移

2.1 实验步骤

2.1.1 任务一：完成鲲鹏代码迁移工具的部署

步骤 1 下载源码。

```
[root@server1 ~]# cd /home
[root@server1 home]# wget -c --no-check-certificate
https://mirror.iscas.ac.cn/kunpeng/archive/Porting_Dependency/Packages/Porting-advisor_2.3.T20_linux-Kunpeng.tar.gz
```

步骤 2 解压源码。

```
[root@server1 home]# tar -zxvf Porting-advisor_2.3.T20_linux-Kunpeng.tar.gz
```

步骤 3 进入目录。

```
[root@server1 home]# cd Porting-advisor_2.3.T20_linux-Kunpeng
```

步骤 4 安装迁移工具 web 模式。

```
[root@server1 Porting-advisor_2.3.T20_linux-Kunpeng]# ./install web
```

在安装过程中，根据提示需要回车或者输入“y”回车。

配置工具安装目录，默认为“/opt”。

配置 web server 的 ip 地址。建议选择与本机一致的 ip 地址。

配置 HTTPS 端口，默认端口 8084。

配置 tool 端口，默认端口 7998。

安装成功后，回显信息如下，可以看到工具最终的 ip 地址和端口：

```
Nginx install success!
Checking for nginx install success!
Encrypting passphase successfully.
It may take a few minutes to generate the certificate, please wait...
Certificate generated successfully. You can import the root certificate to the browser to mask security alarms when you access the tool. The root certif
icate is stored in /opt/portadv/tools/webui/ca.crt.
Created symlink /etc/systemd/system/multi-user.target.wants/nginx_port.service → /usr/lib/systemd/system/nginx_port.service.
Locking password for user porting.
passwd: Success
Created symlink /etc/systemd/system/multi-user.target.wants/gunicorn_port.service → /etc/systemd/system/gunicorn_port.service.
Porting Web console is now running, go to: https://192.168.0.220:8084/porting/#/login
Successfully installed the Kunpeng Porting Advisor in /opt/portadv/.
```

步骤 5 验证。

打开本地浏览器，在浏览器地址栏输入 https://部署服务器的 ip:端口号/porting/#/login

（例如 https://124.71.228.105:8084/porting/#/login），输入用户名（默认用户名 portadmin）和密码（首次登录时需要设置密码），登录成功。

2.1.2 任务二：使用鲲鹏代码迁移工具修改源码

2.1.2.1 扫描源码

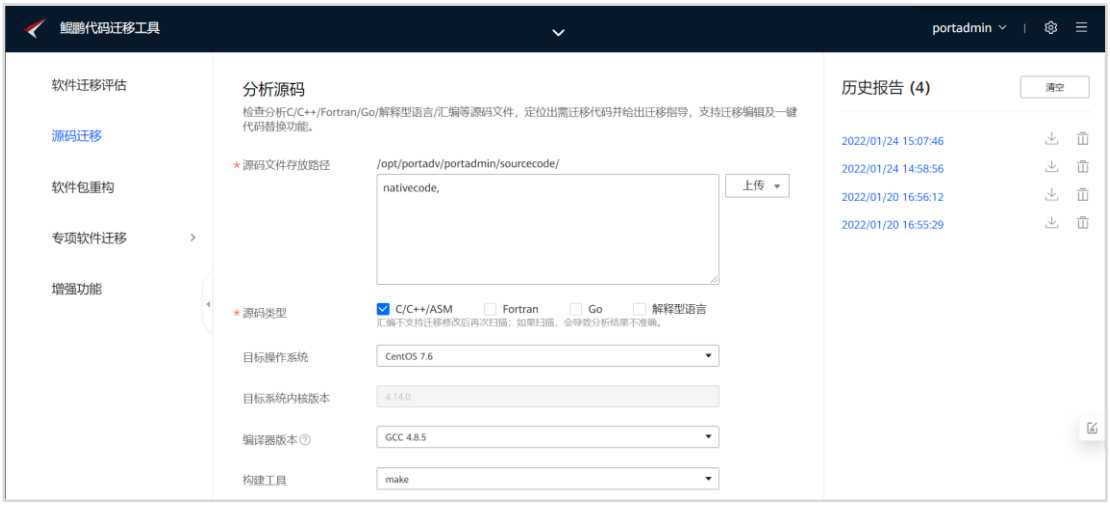
步骤 1 从 codehub 下载源码，并放置源码包到迁移工具，设置参数。

```
[root@server1 ~]# cd /home/
[root@server1 home]# git clone https://codehub.devcloud.cn-north-4.huaweicloud.com/kpcxsj00002/IMSystem.git
[root@server1 home]# cd /home/IMSystem/src/main/resources
[root@server1 resources]# cp -r nativecode/ /opt/portadv/portadmin/sourcecode/nativecode
[root@server1 resources]# cd /opt/portadv/portadmin/sourcecode/
[root@server1 sourcecode]# chown -R porting:porting nativecode/
```

步骤 2 登录鲲鹏代码迁移工具 Web 界面。

打开本地浏览器，在浏览器地址栏输入 https://部署服务器的 ip:端口号/porting/#/login。

（例如 https://124.71.228.105:8084/porting/#/login），输入用户名（默认用户名 portadmin）和密码，登录成功，选择“源码迁移”标签，源码文件存放路径选择 nativecode 目录，参照图中参数配置任务。

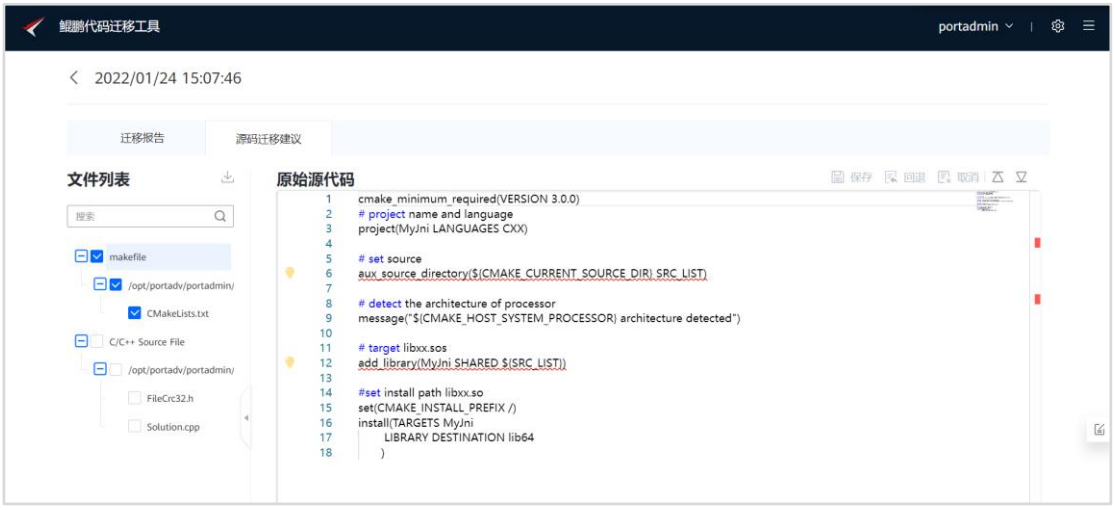


步骤 3 开始分析，查看分析报告。

单击“开始分析”，分析完成后自动弹出如下图所示的分析报告，单击相应的历史报告。

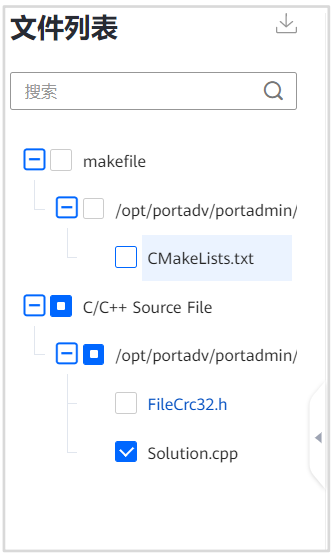
步骤 4 进入“源码迁移建议”页签查看具体的修改建议。

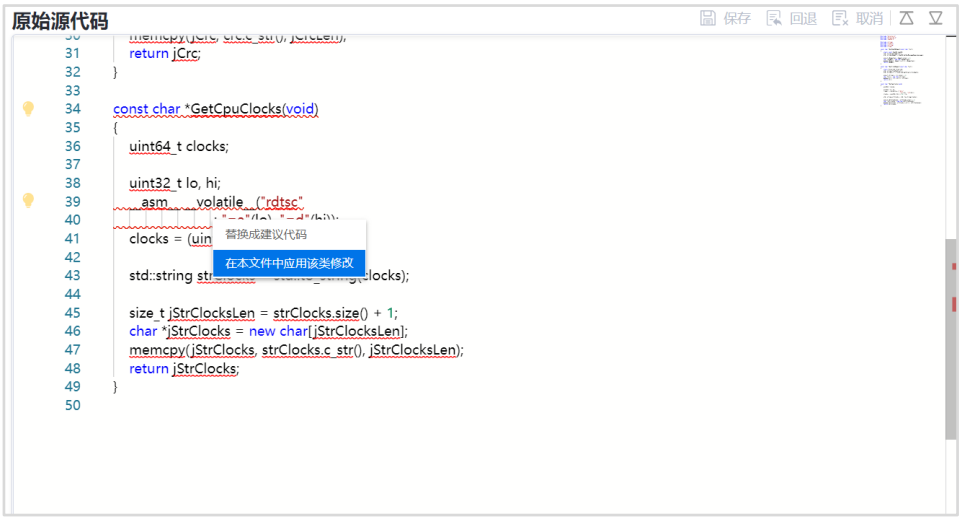
文件列表中显示了需要迁移的文件，红色波浪线标记了需要迁移的代码段。



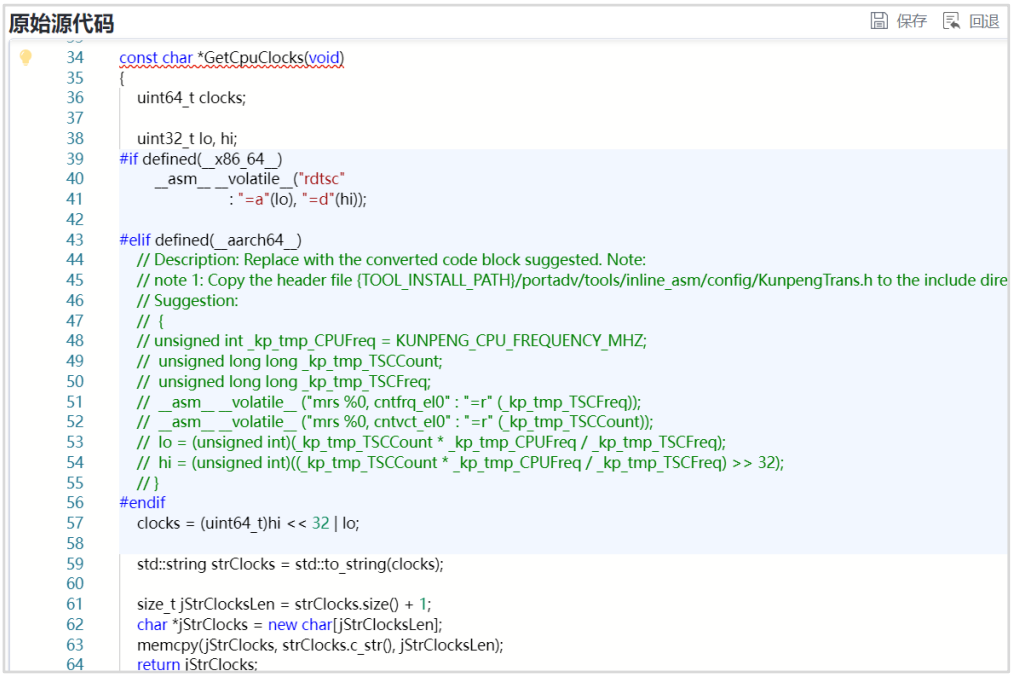
步骤 5 修改 Solution.cpp 代码。

选择 Solution.cpp 文件，鼠标悬浮在需要修改的代码上，单击修改建议，单击“Quick FIX”，再选择“在文本中应用该类修改”。





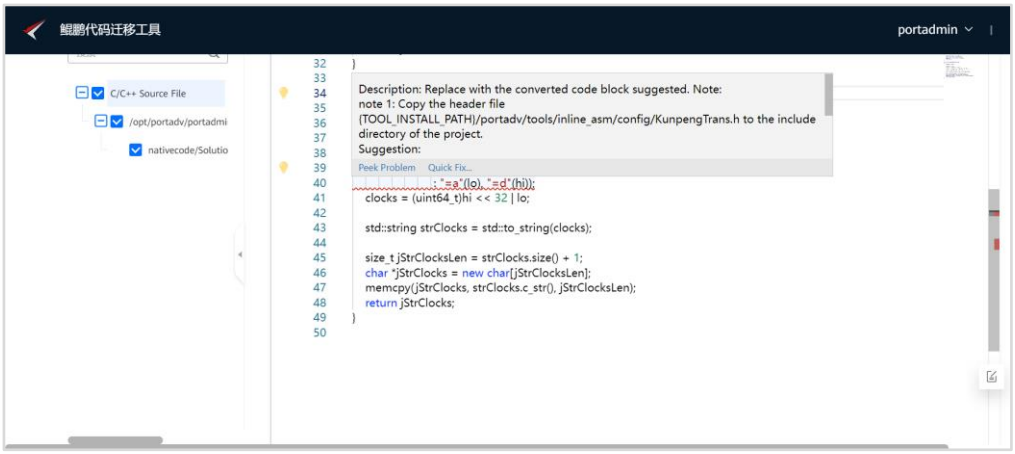
出现如下图所示的修改建议。



理解修改建议后，按照正确的语法修改代码，并保存。

2.1.2.2 添加头文件

根据修改建议中出现的 note1:Copy the header file。将 KunpengTrans.h 复制到 nativecodde 目录下。

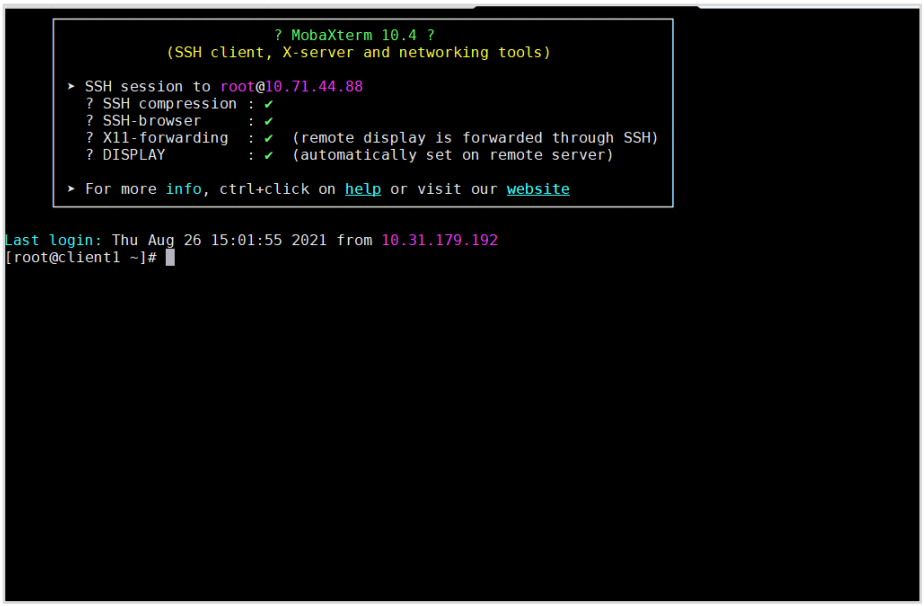


注意：需要自己填写！ cp /opt/portadv/tools/inline_asm/config/KunpengTrans.h /opt/portadv/portadmin/sourcecode/nativecode

2.1.3 任务三：编译代码生成动态库

步骤 1 登录远程服务器。

通过远程登录工具如 CloudShell、MobaXterm、SSH、putty 等工具登录到远程服务器。



步骤 2 进入到代码所在目录。

```
[root@server1 ~]# cd    /opt/portadv/portadmin/sourcecode/nativecode
```

步骤 3 查看当前目录文件。

```
[root@server1 nativecode]# ll
```

可以看到当前目录下的所有文件，注意到被修改的文件 solution.cpp 已经有了备份*.bak.0。

```
[root@server1 nativecode]# ll
total 40K
-rw-------+ 1 porting porting  468 Jan 24 15:06 CMakeLists.txt
-rw-------+ 1 porting porting 1.3K Jan 24 15:06 FileCrc32.cpp
-rw-------+ 1 porting porting  956 Jan 24 15:31 FileCrc32.h
-rw-------  1 porting porting  875 Jan 24 15:31 FileCrc32.h.20220124150746.bak.0
-rw-------+ 1 porting porting 5.2K Jan 24 15:06 Sha256.cpp
-rw-------+ 1 porting porting 1.9K Jan 24 15:06 Sha256.h
-rw-------+ 1 porting porting 2.0K Jan 24 15:51 Solution.cpp
-rw-------  1 porting porting 1.3K Jan 24 15:51 Solution.cpp.20220124150746.bak.0
-rw-------+ 1 porting porting  278 Jan 24 15:06 Solution.h
```

步骤 4 新建 build 目录。

在命令行中依次输入：

```
[root@server1 nativecode]# mkdir build
[root@server1 nativecode]# cd build/
[root@server1 build]# pwd
```

```
[root@server1 build]# pwd
/opt/portadv/portadmin/sourcecode/nativecode/build
```

步骤 5 尝试编译。

在命令行中输入如下内容：（如果没有 cmake，需要 yum install cmake）

```
[root@server1 build]# cmake ..
[root@server1 build]# make
[root@server1 build]# make install
```

发现在 make 时，编译出现如下错误：

```
[root@server1 build]# make
Scanning dependencies of target MyJni
[ 25%] Building CXX object CMakeFiles/MyJni.dir/FileCrc32.cpp.o
/tmp/ccW9Oydn.s: Assembler messages:
/tmp/ccW9Oydn.s:364: Error: selected processor does not support `crc32cb w0,w0,w1'
make[2]: *** [CMakeFiles/MyJni.dir/build.make:63: CMakeFiles/MyJni.dir/FileCrc32.cpp.o] Error 1
make[1]: *** [CMakeFiles/Makefile2:73: CMakeFiles/MyJni.dir/all] Error 2
make: *** [Makefile:130: all] Error 2
```

这是因为 crc32 硬件加速指令在 arm 平台编译时需要添加编译选项“--march=armv8-a+crc”，这时候就需要修改 cmake 配置文件。

步骤 6 修改 cmake 配置文件。

在命令行中输入：

```
[root@server1 build]# vim ../CMakeLists.txt
```

切换到小写输入模式，可以用“h、j、k、l”或者方向键移动光标，移动到如下图所示位置，按“o”键再下方添加新行，进入编辑模式。

```
cmake_minimum_required(VERSION 3.0.0)
# project name and language
project(MyJni LANGUAGES CXX)

# set source
aux_source_directory(${CMAKE_CURRENT_SOURCE_DIR} SRC_LIST)

# detect the architecture of processor
message("${CMAKE_HOST_SYSTEM_PROCESSOR} architecture detected")

# target libxx.sos
add_library(MyJni SHARED ${SRC_LIST})


#set install path libxx.so
set(CMAKE_INSTALL_PREFIX /)
install(TARGETS MyJni
        LIBRARY DESTINATION lib64
       )
```

输入如下内容：

```
if(CMAKE_HOST_SYSTEM_PROCESSOR MATCHES "aarch64")
add_compile_options(-march=armv8-a+crc)
endif()
```

编辑完成后，按“ESC”键退出编辑模式，再依次输入“:”、“w”、“q”,最后回车保存并退出文档。文档可以在本地 pc 编辑后上传到原目录。

```
cmake_minimum_required(VERSION 3.0.0)
# project name and language
project(MyJni LANGUAGES CXX)

# set source
aux_source_directory(${CMAKE_CURRENT_SOURCE_DIR} SRC_LIST)

# detect the architecture of processor
message("${CMAKE_HOST_SYSTEM_PROCESSOR} architecture detected")
if(CMAKE_HOST_SYSTEM_PROCESSOR MATCHES "aarch64")
    add_compile_options(-march=armv8-a+crc)
endif()

# target libxx.sos
add_library(MyJni SHARED ${SRC_LIST})


#set install path libxx.so
set(CMAKE_INSTALL_PREFIX /)
install(TARGETS MyJni
        LIBRARY DESTINATION lib64
       )
```

步骤 7 再次编译。

在命令行中输入：

```
[root@server1 build]# cd /opt/portadv/portadmin/sourcecode/nativecode/build
[root@server1 build]# rm -rf *
[root@server1 build]# cmake ..
[root@server1 build]# make
[root@server1 build]# make install
```

注意：这一步请确保工作目录是 build 文件夹，可以通过 pwd 命令查看。

清除之前的编译中间文件后，重新按照步骤 5 执行，这时候编译过程没有问题，输出内容如下图所示。

```
[root@server1 build]# cmake ..
-- The CXX compiler identification is GNU 7.3.0
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
aarch64 architecture detected
-- Configuring done
-- Generating done
-- Build files have been written to: /opt/portadv/portadmin/sourcecode/nativecode/build
[root@server1 build]# make
Scanning dependencies of target MyJni
[ 25%] Building CXX object CMakeFiles/MyJni.dir/FileCrc32.cpp.o
[ 50%] Building CXX object CMakeFiles/MyJni.dir/Sha256.cpp.o
[ 75%] Building CXX object CMakeFiles/MyJni.dir/Solution.cpp.o
[100%] Linking CXX shared library libMyJni.so
[100%] Built target MyJni
[root@server1 build]# make install
[100%] Built target MyJni
Install the project...
-- Install configuration: ""
-- Installing: /lib64/libMyJni.so
```

步骤 8 查看编译安装结果。

上图红框内可知，动态库 libMyJni.so 被安装到了/lib64 目录下，可以通过如下命令查验。

在命令行中输入：

```
[root@server1 build]# ll /lib64/libMyJni.so
```

从下图可以看到，动态库文件已经在在目标目录中了，至此编译安装结束，关卡完成。

```
[root@server1 build]# ll /lib64/libMyJni.so
-rwxr-xr-x 1 root root 176K Jan 24 16:04 /lib64/libMyJni.so
```

2.2 课后拓展

通过鲲鹏开发套件实现软件包迁移

实验地址：https://lab.huaweicloud.com/testdetail_436

2.3 思考题

如果是 JAVA 语言编写的源码，迁移至鲲鹏平台上时修改源码?

参考答案：

JAVA 语言编写的程序，如果源码中未调用编译型语言（如 C/C++）编写的 so 库函数，在 JDK 版本无特殊差异性的情况下，通常不需要修改源码。因为 JAVA 编译生成的.CLASS 文件由 JAVA 虚拟机解释执行。