# 开发环境

Ip：10.19.118.27

Samba jarod/000

Ssh jarod/123

root/123

samba目录说明

/home/common\_api/

Demo代码：[\\10.19.118.27\common\_api\02\_demo](file:///\\10.19.118.27\common_api\02_demo)

各模块开发目录（建议）：[\\10.19.118.27\common\_api\03\_work](file:///\\10.19.118.27\common_api\03_work)

Qemu虚拟机的文件系统目录

\\10.19.118.27\qemu\_rootfs\NFSrootfs

配置环境变量

source /home/common\_api/environment-setup-core2-64-poky-linux

# QEMU环境

Ip：192.168.7.2

Ssh：root/root（通过118.27连ssh root@192.168.7.2）

对应文件系统目录：

/home/wangqi/workspace/rootfs/NFSrootfs/

# 编写fidl文件

参考它http://10.19.118.120/svn/RAD/Shared/01.SystemSupport/common\_api\_demo/demo.fidl

从它中选择自己需要的

**编写fidl文件头的部分必须按照如下格式并保证关键字（package interface major minor）的唯一性**

package APP\_NAME  **独立一行**

interface HelloWorld **独立一行，大括号也不能在这行**

{

version

{

major 88  **独立一行，大括号也不能在这行**

minor 99 **独立一行，大括号也不能在这行**

}

注：关键字**精确匹配**（package和packages可辨别），前后多个空格没有问题

参考官方文档

http://10.19.118.120/svn/RAD/Shared/01.SystemSupport/common\_api\_depends/附录：FrancaUserGuide-0.9.2.0.pdf

# 取CmakeList.txt到工作目录

准备工作：创建工作目录（路径）

<http://10.19.118.120/svn/RAD/Shared/01.SystemSupport/common_api_demo/CMakeLists.txt.bak>

# 编译

http://10.19.118.120/svn/RAD/Shared/01.SystemSupport/common\_api\_demo/auto.sh

脚本说明

1. 功能：编译头文件、lib动态库并拷贝到指定目录
2. 在工作目录内创建fidl文件和cmake文件平行放置
3. 取脚本文件(auto.sh)和fidl文件(demo里为demo.fidl)、cmake文件（CmakeList.txt）一起放在工作目录并执行脚本（3个文件保证有且仅有1个）
4. 脚本与fidl文件协同参数即编写fidl文件说明：

共计4个参数（与fidl文件定义一致）

package major minor interface等四个必要变量sh脚本自动从fidl文件获取

1. 脚本参数说明

支持一个参数：clean

Clean的时候清除库及其相关的目录和文件、清除app\_service/include/和app\_client/include/

1. 结果文件

工作目录内生成：库文件（.so）、app\_service和app\_client目录（内均有include目录）。库文件和对应include内的头文件供app开发使用（app\_service/include/目录供service开发使用app\_client/include/目录供client开发使用）

# 创建应用

## 4.1、创建客户端应用

客户端示例代码：

|  |
| --- |
| #include <CommonAPI/CommonAPI.hpp>  #include <v0\_1/APP\_NAME/HelloWorldProxy.hpp>  using namespace v0\_1::APP\_NAME;  int main() {  /\* LogContext：如果使用DLT作为LOG输出手段，则本参数生效。 \*/  /\* 本参数用于设置DLT的LOG输出时使用的Context值，是一个最大4字节的字符串。\*/  /\* 如果不进行设置，默认值是"CAPI"。 \*/  CommonAPI::Runtime::setProperty("LogContext", "DEMC");  /\* LibraryBase：用于设置binding-library的名称。 \*/  /\* 例如设置本参数为Abc并指定Binding类型为"dbus" \*/  /\* 则CommonAPI自动寻找一个叫做"libAbc-dbus.so"的库文件进行载入 \*/  /\* 如下定义则自动寻找一个叫做"libAPP\_NAME-dbus.so"的库文件 \*/  CommonAPI::Runtime::setProperty("LibraryBase", "APP\_NAME");  /\* 空 \*/  std::string domain = "";  std::string instance = "";  std::string connection = "";  /\* 初始化 \*/  std::shared\_ptr < CommonAPI::Runtime > runtime = CommonAPI::Runtime::get();  std::shared\_ptr<HelloWorldProxy<>> myProxy = runtime->buildProxy<HelloWorldProxy>(domain,instance, connection);  return 0;  } |

## 4.2、创建服务端应用

服务端示例代码：

|  |
| --- |
| #include <CommonAPI/CommonAPI.hpp>  #include "APP\_NAMEStubImpl.hpp"  using namespace std;  int main() {  /\* LogContext：如果使用DLT作为LOG输出手段，则本参数生效。 \*/  /\* 本参数用于设置DLT的LOG输出时使用的Context值，是一个最大4字节的字符串。\*/  /\* 如果不进行设置，默认值是"CAPI"。 \*/  CommonAPI::Runtime::setProperty("LogContext", "DEMS");  /\* LibraryBase：用于设置binding-library的名称。 \*/  /\* 例如设置本参数为Abc并指定Binding类型为"dbus" \*/  /\* 则CommonAPI自动寻找一个叫做"libAbc-dbus.so"的库文件进行载入 \*/  /\* 如下定义则自动寻找一个叫做"libAPP\_NAME-dbus.so"的库文件 \*/  CommonAPI::Runtime::setProperty("LibraryBase", "APP\_NAME");  /\* 空 \*/  std::string domain = "";  std::string instance = "";  std::string connection = "";  /\* 初始化 \*/  std::shared\_ptr<CommonAPI::Runtime> runtime = CommonAPI::Runtime::get();  std::shared\_ptr<HelloWorldStubImpl> myService = std::make\_shared<HelloWorldStubImpl>();  bool successfullyRegistered = runtime->registerService(domain, instance, myService, connection);  return 0;  } |

# Interface Definition （Fidl配置文件选项说明）

## 5.1、Method

CommonAPI C++的IPC接口使用的是通用的“服务端、客户端”模式。

对于客户端主动发起的通讯，采取的是方法（Methods）调用的方式来实现。

Service

Client

**Methods**

Method分为两种类型：

标准的带有返回值的Method（Basic Method）

不需要返回值的Method（Fire-and-forgot Method）

## 5.2、Broadcasts（signal）

服务端主动发起的通讯通过广播（Broadcasts）的方式实现。

Service

Client1

Client2

**Broadcasts**

需要注意的是，CommonAPI C++提供的广播模式也有两种：

标准的广播方式（Basic Broadcasts），即广播内容发送给所有的客户端

选择性广播方式（Selective Broadcasts），即广播内容可以发送给某一个或一组客户端。此种情况下，通常是由服务端提供类似Registor的Method用于客户端注册。然后服务端根据注册情况将广播发送给不同的客户端。（参照 [E04PhoneBook](http://git.projects.genivi.org/?p=ipc/common-api-tools.git;a=tree;f=CommonAPI-Examples/E04PhoneBook) ）

## 5.3、sayHello case

|  |
| --- |
| <\*\* @description : 这是一个标准的函数接口，拥有一个输入参数，一个输出参数 \*\*>  method sayHello {  in {  String name  }  out {  String message  }  } |

Client端

|  |
| --- |
| const std::string name = "World";  CommonAPI::CallStatus callStatus;  std::string returnMessage;  CommonAPI::CallInfo info(1000);  myProxy->sayHello(name, callStatus, returnMessage, &info);  if (callStatus != CommonAPI::CallStatus::SUCCESS) {  std::cerr << "sayHello:Remote call failed!\n";  return -1;  }  std::cout << "Got message: '" << returnMessage << "'\n"; |

Service端

|  |
| --- |
| void HelloWorldStubImpl::sayHello(const std::shared\_ptr<CommonAPI::ClientId> \_client,std::string \_name,sayHelloReply\_t \_reply) {  std::stringstream messageStream;  messageStream << "Hello " << \_name << "================" << "!";  \_reply(messageStream.str());  }; |

# Refrences

http://10.19.118.120/svn/RAD/Shared/01.SystemSupport/common\_api\_depends/FAW HS7 IVI System Design IPC-CAPI.docx

# else

ldd命令

ldd命令通常使用"-v"或"--verbose"选项来显示所依赖的动态连接库的尽可能的详细信息。

切换root用户

sudo bash