

[主页](#) [博客](#) [相册](#) [个人档案](#) [好友](#)

天道酬勤

[查看文章](#)

Borland VisiBroker for C++ 开发实例

2009-06-14 11:00

Tony Tian

本文说明使用 **Borland VisiBroker** 开发**CORBA** 应用的过程。供 **Visibroker** 入门使用者参考。例程可以在 **VisiBroker** 安装目录下的**examples\basic\bank_agent** 下找到，**VisiBroker** 下载地址是：<http://www.borland.com/visibroker/download/>。

开发过程遵循以下几个步骤：

1. 为 **CORBA** 应用中的每个对象定义**IDL**接口。

在本例中定义两个**IDL**接口，一个名为 **Account**，内含方法：**balance()**。一个名为**AccountManager**，内含方法 **open()**。

2. 编译 **IDL** 文件。

使用 **VisiBroker IDL Compiler - idl2cpp** 编译 **IDL**文件，生成客户端的 **stub**代码和服务端端的 **POA servant** 代码。

3. 编写客户端程序代码。

初始化**ORB**，绑定 **Account** 对象和 **AccountManager** 对象，并调用相关方法。

4. 编写服务器端程序代码。

初始化**ORB**，提供相关接口方法的实现。

5. 用 **C++** 编译器编译客户端程序和服务器端程序。

6. 运行服务器程序。

7. 运行客户端程序。

下面我们分别讨论各个步骤所作的工作。

一. 定义IDL接口

IDL 语言用于描述一个 **CORBA** 对象所提供的服务，以及如何调用这些服务。本例定义名为 **bank.idl** 的 **IDL** 接口文件。内容如下所示：

```
module Bank{
interface Account {
float balance()
}
interface AccountManage {
Account open(in tring name)
}
}
```

二. 编译 IDL 文件

我们使用下列方式编译**bank.idl**文件：

```
prompt>idl2cpp bank.idl
```

编译后，生成了如下文件：

Bank_c.hh：包含 **Account** 和 **AccountManager** 类的定义。

Bank_c.cpp：包含供客户端使用的**stub** 接口程序。

Bank_s.hh：包含 **AccountPOA** 和 **AccountManagerPOA servant** 类的定义。

Bank_s.cpp：包含供服务器端使用的接口程序。

三. 编写客户端程序代码

客户端代码(**Client.c**)完成**ORB**初始化，绑定 **Account** 和 **AccountManager** 对象，并调用相关方法。程序如下：

```
#include "Bank_c.hh"
int main(int argc, char* const* argv)
{
try {
// Initialize the ORB.
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

// Get the manager Id
PortableServer::ObjectId_var managerId =
PortableServer::string_to_ObjectId("BankManager");

// Locate an account manager. Give the full POA name and the servant ID.
Bank::AccountManager_var manager =
Bank::AccountManager::_bind("/bank_agent_poa", managerId);

// use argv[1] as the account name, or a default.
const char* name = argc > 1 ? argv[1] : "Jack B. Quick";
```

```

// Request the account manager to open a named account.
Bank::Account_var account = manager->open(name);

// Get the balance of the account.
CORBA::Float balance = account->balance();

// Print out the balance.
cout << "The balance in " << name << "'s account is $" << balance << endl;
}
catch(const CORBA::Exception& e) {
cerr << e << endl;
return 1;
}

return 0;
}

```

四. 编写服务器端程序代码

服务器端代码(**Server.c**)完成 **ORB** 和 **POA** 初始化, 提供相关接口方法的实现, 并完成服务器程序的 **main** 函数, 程序如下:

```

#include "BankImpl.h"

int main(int argc, char* const* argv)
{
try {
// Initialize the ORB.
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

// get a reference to the root POA
CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(obj);

CORBA::PolicyList policies;
policies.length(1);
policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);

// get the POA Manager
PortableServer::POAManager_var poa_manager = rootPOA->the_POAManager();

// Create myPOA with the right policies
PortableServer::POA_var myPOA = rootPOA->create_POA("bank_agent_poa", poa_manager,policies);
// Create the servant
AccountManagerImpl managerServant;

// Decide on the ID for the servant
PortableServer::ObjectId_var managerId =PortableServer::string_to_ObjectId("BankManager");

// Activate the servant with the ID on myPOA
myPOA->activate_object_with_id(managerId, &managerServant);

```

```

// Activate the POA Manager
poa_manager->activate();

CORBA::Object_var reference = myPOA->servant_to_reference(&managerServant);
cout << reference << " is ready" << endl;

// Wait for incoming requests
orb->run();
}
catch(const CORBA::Exception& e) {
cerr << e << endl;
return 1;
}
return 0;
}

```

其中**BankImpl.h**的部分程序如下：

```

#include "Bank_s.hh"
.....
class AccountImpl : public virtual POA_Bank::Account,
public virtual PortableServer::RefCountServantBase
{ public:
AccountImpl(CORBA::Float balance) : _balance(balance)
{}
CORBA::Float balance() { return _balance; }
private:
CORBA::Float _balance;
};

class AccountManagerImpl : public POA_Bank::AccountManager
{ public:
AccountManagerImpl() {}
Bank::Account_ptr open(const char* name) {
// Lookup the account in the account dictionary.
PortableServer::ServantBase_var servant = _accounts.get(name);

if (servant == PortableServer::ServantBase::_nil()) {
// Make up the account's balance, between 0 and 1000 dollars.
CORBA::Float balance = abs(rand()) % 100000 / 100.0;
// Create the account implementation, given the balance.
servant = new AccountImpl(balance);
// Print out the new account
cout << "Created " << name << "'s account." << endl;
// Save the account in the account dictionary.
_accounts.put(name, servant);
}
try {
// Activate it on the default POA which is root POA for this servant
PortableServer::POA_var default_poa = _default_POA();
CORBA::Object_var ref = default_poa->servant_to_reference(servant);

```

```

Bank::Account_var account = Bank::Account::_narrow(ref);

// Print out the new account
cout << "Returning " << name << "'s account: " << account << endl;

// Return the account
return Bank::Account::_duplicate(account);
}
catch(const CORBA::Exception& e) {
cerr << "_narrow caught exception: " << e << endl;
}

return Bank::Account::_nil();
}
};

```

五. 编译客户端程序和服务器端程序

我们需要将 **stub** 代码和客户端程序一起编译成客户端程序。将服务器代码和生成的 **Servent** 代码一起编译生成服务器端程序。假设 **VisiBroker** 安装在 **C:\vbroker**，我们使用 **Visual C++** 编译器来编译我们的例子。方式如下：

```

prompt> cd vbroker\example \basic\bank_agent
prompt> nmake -f Makefile.cpp

```

其中，**Makefile.cpp** 文件内容如下：

```

include .././stdmk_nt

EXE = Client.exe Server.exe

all: $(EXE)

Bank_c.cpp: Bank.idl
$(ORBCC) Bank.idl

Bank_s.cpp: Bank.idl
$(ORBCC) Bank.idl

Client.exe: Bank_c.obj Client.obj
$(LINK_EXE) /out:Client.exe Client.obj \
Bank_c.obj $(LIBORB) $(STDCC_LIBS)

Server.exe: Bank_s.obj Bank_c.obj Server.obj
$(LINK_EXE) /out:Server.exe Server.obj \
Bank_s.obj Bank_c.obj $(LIBORB) $(STDCC_LIBS)

```

其中：文件 **stdmk_nt** 内容如下：

```

#####
### Windows Visual C++ (nmake) definitions
#####

```

```

!IFDEF VBROKERDIR
!MESSAGE Warning: VBROKERDIR is not set
!MESSAGE
!ENDIF

!IF "$ (BUILD_TYPE)" == "debug" || "$ (BUILD_TYPE)" == "DEBUG"
DEBUG = 1
!ENDIF

!IF "$ (STD)" == "1"
STL = 1
!ENDIF

!IFDEF DEBUG
!IFDEF STL
CC_MODEL = /MDd /DTHREAD
!ELSE
CC_MODEL = /MDd /DTHREAD /D_VIS_STD
!ENDIF
DEBUG = /Zi
LDFLAGS = /DEBUG:FULL
!ELSE
!IFDEF STL
CC_MODEL = /MD /DTHREAD
!ELSE
CC_MODEL = /MD /DTHREAD /D_VIS_STD
!ENDIF
DEBUG =
LDFLAGS =
!ENDIF

### Platform specific compiler definitions
CC = CL /nologo $(CC_MODEL) -DWIN32 /GX /DSTRICT /DALIGNED
CC_EXE_TARGET = /Fo
LINK_DLL = LINK /nologo $(LDFLAGS) /DLL
LINK_DLL_TARGET = /OUT:
LINK_EXE = LINK $(LDFLAGS)
LINK_EXE_TARGET = /OUT:
STDCC_LIBS =
CCDEFS =

### VisiBroker directory locations
ORBCC = $(VBROKERDIR)\bin\idl2cpp -src_suffix cpp
LIBDIR = $(VBROKERDIR)\lib
CCINCLUDES = -I. -I$(VBROKERDIR)\include -I$(VBROKERDIR)\include\stubs

### ORB library
LIBORB = /LIBPATH:$(LIBDIR)

### Compiler flags
CCFLAGS = $(CCINCLUDES) $(DEBUG) $(CCDEFS)

```

```
### Standard build rules for .cpp files

.SUFFIXES: .CPP .C .obj .h .hh .java .class

.C.obj:
$(CC) $(CCFLAGS) -c /Tp $<

.CPP.obj:
$(CC) $(CCFLAGS) -c /Tp $<
```

六. 运行服务器程序和客户端程序

在运行程序之间需要在网络中运行一个 **VisiBroker Smart Agent** , 然后分别启动相应服务器程序和客户程序:

```
prompt> start -c osagent
prompt> start Server
prompt> Client
```

下面是客户端的输出结果:

The balance in the account in \$168.38.

至此, 一个 **VisiBroker** 应用程序开发运行成功。

Tony Tian (t111@263.net)


类别: Corba | 转帖 + | 添加到收藏 | 分享到i贴吧 | 浏览(195) | 评论 (0)

上一篇: [corba与visibroker](#) 下一篇: [CORBA IDL 部分语法](#)


相关文章:

- Borland Enterprise Server v6.5...
- Borland VisiBroker Smart Agent...


最近读者:



登录后, 您就出现在这里。



[micheals23](#)



[406983551](#)

网友评论:
发表评论:

姓 名:

注册 | 登录

网址或邮箱:

(选填)

:

内 容:

插入表情

验证码:

请点击后输入四位验证码，字母不区分大小写

©2010 Baidu