RMI z IIOP POA (do współpracy z omniORB)

Tomasz Kubik

Java

Napisz interfejs:

HelloInterface.java

```
import java.rmi.Remote;

public interface HelloInterface extends java.rmi.Remote {
  public void sayHello() throws java.rmi.RemoteException;
}
```

Napisz implementację:

HelloImpl.java

```
import javax.rmi.PortableRemoteObject;

public class HelloImpl extends PortableRemoteObject implements HelloInterface {
  public HelloImpl() throws java.rmi.RemoteException {
    super();     // invoke rmi linking and remote object initialization
  }

  public void sayHello() throws java.rmi.RemoteException {
    System.out.println( "It works!  Hello World!!" );
  }
}
```

Napisz klienta:

HelloClient.java

```
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import javax.rmi.*;
import java.util.Vector;
import javax.naming.NamingException;
import javax.naming.InitialContext;
import javax.naming.Context;

public class HelloClient {

  public static void  main( String args[] ) {
    Context ic;
    Object objref;
    HelloInterface hi;

    try {
      ic = new InitialContext();
    } catch (NamingException e) {
       System.out.println("failed to obtain context" + e);
       e.printStackTrace();
       return;
    }
```

```
    // STEP 1: Get the Object reference from the Name Service
    // using JNDI call.
    try {
      objref = ic.lookup("HelloService");
      System.out.println("Client: Obtained a ref. to Hello server.");
    } catch (NamingException e) {
        System.out.println("failed to lookup object reference");
        e.printStackTrace();
        return;
    }

    // STEP 2: Narrow the object reference to the concrete type and
    // invoke the method.
    try {
      hi = (HelloInterface) PortableRemoteObject.narrow(
        objref, HelloInterface.class);
      hi.sayHello();
    } catch (ClassCastException e) {
        System.out.println("narrow failed");
        e.printStackTrace();
        return;
      } catch( Exception e ) {
            System.err.println( "Exception " + e + "Caught" );
            e.printStackTrace( );
            return;
        }
  }
}
```

Napisz serwer:

HelloServer.java

```
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.rmi.PortableRemoteObject ;
import com.sun.corba.se.internal.POA.POAORB;
import org.omg.PortableServer.*;
import java.util.*;
import org.omg.CORBA.*;
import javax.rmi.CORBA.Stub;
import javax.rmi.CORBA.Util;

public class HelloServer {
  public HelloServer(String[] args) {
    try {
      Properties p = System.getProperties();
      // add runtime properties here
      p.put("org.omg.CORBA.ORBClass",
          "com.sun.corba.se.internal.POA.POAORB");
      p.put("org.omg.CORBA.ORBSingletonClass",
          "com.sun.corba.se.internal.corba.ORBSingleton");

      ORB orb = ORB.init( args, p );

      POA rootPOA = (POA)orb.resolve_initial_references("RootPOA");

      // STEP 1: Create a POA with the appropriate policies
      Policy[] tpolicy = new Policy[3];
      tpolicy[0] = rootPOA.create_lifespan_policy(
        LifespanPolicyValue.TRANSIENT );
```

```
    tpolicy[1] = rootPOA.create_request_processing_policy(
      RequestProcessingPolicyValue.USE_ACTIVE_OBJECT_MAP_ONLY );
    tpolicy[2] = rootPOA.create_servant_retention_policy(
      ServantRetentionPolicyValue.RETAIN);
    POA tPOA = rootPOA.create_POA("MyTransientPOA", null, tpolicy);

    // STEP 2: Activate the POA Manager, otherwise all calls to the
    // servant hang because, by default, POAManager will be in the
    // HOLD state.
    tPOA.the_POAManager().activate();

    // STEP 3: Instantiate the Servant and activate the Tie, If the
    // POA policy is USE_ACTIVE_OBJECT_MAP_ONLY
    HelloImpl helloImpl = new HelloImpl();
    _HelloImpl_Tie tie = (_HelloImpl_Tie)Util.getTie( helloImpl );
    String helloId = "hello";
    byte[] id = helloId.getBytes();
    tPOA.activate_object_with_id( id, tie );


    // STEP 4: Publish the object reference using the same object id
    // used to activate the Tie object.
    Context initialNamingContext = new InitialContext();
    initialNamingContext.rebind("HelloService",
      tPOA.create_reference_with_id(id,
        tie._all_interfaces(tPOA,id)[0]) );
    System.out.println("Hello Server: Ready...");

    // STEP 5: Get ready to accept requests from the client
    orb.run();
   }
    catch (Exception e) {
      System.out.println("Problem running HelloServer: " + e);
      e.printStackTrace();
    }
  }

  public static void main(String args[]) {
    new HelloServer( args );
  }
}
```

Skompiluj źródła:

	komenda	powstałe pliki
1	javac –d . –classpath . HelloImpl.java	HelloImpl.class
2	rmic –poa –iiop HelloImpl	_HelloInterface_Stub.class, _HelloImpl_Tie.class
3	javac –d . –classpath . HelloInterface.java HelloServer.java HelloClient.java	HelloInterface.class HelloServer.class HelloClient.class

Utwórz plik iterfejsu w języku idl:

	komenda	powstałe pliki
	rmic –idl HelloInterface	HelloInterface.idl

omniORB (4.0.5)

Skompiluj plik interfejsu w języku idl:

	komenda	powstałe pliki
	omniidl –Ie:\omniORB-4.0.5\idl –bcxx –Wbtp –Wbexample –Wbh=.h –Wbs=.cpp HelloInterface.idl	HelloInterface_i.cc HelloInterface.h HelloInterface.cpp

HelloInterface_i.cc - przykładowy plik (implementacja serwera)
HelloInterface.h – nagłówek
HelloInterface.cpp –szkielet

Napisz klienta

HelloInterface_client.cpp

```
#include <iostream.h>
#include "HelloInterface.h"

static CORBA::Object_ptr getObjectReference(CORBA::ORB_ptr orb);


int main (int argc, char **argv) {
  try {
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);//, "omniORB3");

   // receive reference from NameService
    CORBA::Object_var obj = getObjectReference(orb);

    HelloInterface_var helloInterfaceref = HelloInterface::_narrow(obj);
    if( CORBA::is_nil(helloInterfaceref) ) {
                        cerr << "hello: The object reference is nil!\n" << endl;
                        return -1; }
    helloInterfaceref->sayHello();
    cerr << "I did: sayHello()" << endl;
    orb->destroy();
  }
  catch(CORBA::COMM_FAILURE& ex) {
    cerr << "Caught system exception COMM_FAILURE -- unable to "
         << "contact the object." << endl;
  }
  catch(CORBA::SystemException&) {
    cerr << "Caught CORBA::SystemException." << endl;
  }
  catch(CORBA::Exception&) {
    cerr << "Caught CORBA::Exception." << endl;
  }
  catch(omniORB::fatalException& fe) {
    cerr << "Caught omniORB::fatalException:" << endl;
    cerr << "  file: " << fe.file() << endl;
    cerr << "  line: " << fe.line() << endl;
    cerr << "  mesg: " << fe.errmsg() << endl;
  }
  catch(...) {
    cerr << "Caught unknown exception." << endl;
  }
  return 0;
}

static CORBA::Object_ptr
getObjectReference(CORBA::ORB_ptr orb) {
  CosNaming::NamingContext_var rootContext;

  try {
    // Obtain a reference to the root context of the Name service:
```

```
    CORBA::Object_var obj;
    obj = orb->resolve_initial_references("NameService");

    // Narrow the reference returned.
    rootContext = CosNaming::NamingContext::_narrow(obj);
    if( CORBA::is_nil(rootContext) ) {
      cerr << "Failed to narrow the root naming context." << endl;
      return CORBA::Object::_nil();
    }
  }
  catch(CORBA::ORB::InvalidName& ex) {
    // This should not happen!
    cerr << "Service required is invalid [does not exist]." << endl;
    return CORBA::Object::_nil();
  }

  // Create a name object, containing the name test/context:
  CosNaming::Name name;
  name.length(1);

  name[0].id   = CORBA::string_dup ("HelloService");       // string copied
  name[0].kind = CORBA::string_dup ("");

  try {
    // Resolve the name to an object reference.
    return rootContext->resolve(name);
  }
  catch(CosNaming::NamingContext::NotFound& ex) {
    // This exception is thrown if any of the components of the
    // path [contexts or the object] aren't found:
    cerr << "Context not found." << endl;
  }
  catch(CORBA::COMM_FAILURE& ex) {
    cerr << "Caught system exception COMM_FAILURE -- unable to "
         << "contact the naming service." << endl;
  }
  catch(CORBA::SystemException&) {
    cerr << "Caught a CORBA::SystemException while using the "
         << "naming service." << endl;
  }
  return CORBA::Object::_nil();
}
```

Napisz serwera (zmodyfikuj plik HelloInterface_i.cc)

HelloInterface_server

```
#include <iostream.h>
#include "HelloInterface.h"

static CORBA::Boolean
bindObjectToName(CORBA::ORB_ptr,CORBA::Object_ptr);

//
// Example class implementing IDL interface HelloInterface
//
class HelloInterface_i: public POA_HelloInterface,
                public PortableServer::RefCountServantBase {
private:
  // Make sure all instances are built on the heap by making the
  // destructor non-public virtual ~HelloInterface_i();
```

```
public:
  // standard constructor
  HelloInterface_i();
  virtual ~HelloInterface_i();
  // methods corresponding to defined IDL attributes and operations
  void sayHello();
};

// Example implementational code for IDL interface HelloInterface
HelloInterface_i::HelloInterface_i(){
  // add extra constructor code here
}
HelloInterface_i::~HelloInterface_i(){
  // add extra destructor code here
}
//   Methods corresponding to IDL attributes and operations
void HelloInterface_i::sayHello(){
  // insert code here and remove the warning
            cout << "It works!  Hello World!!\n"; cout.flush();
}

// End of example implementational code

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

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

    // We allocate the objects on the heap.  Since these are reference
    // counted objects, they will be deleted by the POA when they are no
    // longer needed.
    HelloInterface_i* myHelloInterface_i = new HelloInterface_i();

    PortableServer::ObjectId_var myHelloInterface_iid = poa->activate_object(myHelloInterface_i);

   CORBA::Object_var myHelloInterface_var = myHelloInterface_i->_this();

   CORBA::String_var sior(orb->object_to_string(myHelloInterface_var.in()));
    cerr << """ << (char*)sior << """ << endl; cerr.flush();

    // Obtain a reference to the object, and register it in the naming service.
   if( !bindObjectToName(orb, myHelloInterface_var.in()) ) return 1;

    myHelloInterface_i->_remove_ref();

    // Obtain a POAManager, and tell the POA to start accepting requests on its objects.
    PortableServer::POAManager_var pman = poa->the_POAManager();
    pman->activate();
    cout << "Server started\n";         cout.flush();

    orb->run();
    orb->destroy();
  }
  catch(CORBA::SystemException&) {
    cerr << "Caught CORBA::SystemException." << endl;
  }
```

```
  catch(CORBA::Exception&) {
    cerr << "Caught CORBA::Exception." << endl;
  }
  catch(omniORB::fatalException& fe) {
    cerr << "Caught omniORB::fatalException:" << endl;
    cerr << "  file: " << fe.file() << endl;
    cerr << "  line: " << fe.line() << endl;
    cerr << "  mesg: " << fe.errmsg() << endl;
  }
  catch(...) {
    cerr << "Caught unknown exception." << endl;
  }

  return 0;
}

static CORBA::Boolean
bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr objref) {
   // Bind object to name service
   CosNaming::NamingContext_var nc;
          try {
               CORBA::Object_var _obj1= orb->resolve_initial_references("NameService");
               // narrow this to the naming context
                nc = CosNaming::NamingContext::_narrow(_obj1.in());

               if( CORBA::is_nil(nc) ) {
                              cerr << "Failed to narrow the root naming context." << endl;
                              return 0;
                    }
          }
          catch(CORBA::ORB::InvalidName& ex) {
                    // This should not happen!
                    cerr << "Service required is invalid [does not exist]." << endl;
                    return 0;
          }

   // Bind to CORBA name service. Same name to be requested by client.
   CosNaming::Name name;
   name.length(1);
   name[0].id=CORBA::string_dup("HelloService");
          try {
             nc->rebind (name,objref);
          }
          catch(CORBA::COMM_FAILURE& ex) {
             cerr << "Caught system exception COMM_FAILURE -- unable to contact the "
                       << "naming service." << endl;
                    return 0;
          }
          catch(CORBA::SystemException&) {
                    cerr << "Caught a CORBA::SystemException while using the naming service."
                    << endl;
                    return 0;
          }

          return 1;
}
```

Skompiluj klienta i serwera z biblioteką OmniORB.

Dla VC++ 5/6 skonfiguruj projekt dla serwera (podobnie będzie dla klienta):

a) Dodaj do projektu pliki wygenerowane przez kompilator omniidl oraz źródło serwera
(Insert->"Files into Project"; HelloInterface.h, HelloInterface.cpp, HelloInterface_server.cpp)
b) Ustaw ścieżki dostępu do plików nagłówkowych i bibliotek:

1. Wybierz menu Tools->Options i zakładkę Directories.
2. W polu "Show directories for" wybierz "Include files".
3. Dodaj ścieżkę dostępu do katalogu include w zainstalowanej dystrybucji omniORB (tj. <Top-Level Directory>\include, gdzie <Top-Level Directory> to katalog główny omniORB).
4. W polu "Show directories for" wybierz "Library files".
5. Dodaj ścieżkę dostępu do katalogu bibliotek w dystrybucji omniORB (tj. <Top-Level Directory\lib\x86_win32).

c) Ustaw makra i biblioteki w projekcie:

1. Wybierz menu Build->Settings i zakładkę "C/C++".
2. W polu "Category" wybierz "C++ Language". Zaznacz "Enable exception handling".
3. W polu "Category" wybierz "Code Generation". W polu "Use run-time library" wybierz "Multithreaded DLL".
4. W polu "Category" wybierz "Preprocessor". Następnie w polu "Preprocessor" dodaj makra:
__WIN32__,__x86__,_WIN32_WINNT=0x0400

   Jeśli jest to NT 4.0, Windows 2000, XP, Windows 2003 Server, dodaj makra:
__NT__ , __OSVERSION__=4

5. Wybierz zakładlę "Link"
6. W polu "Category" wybierz "Input". W polu "Object/library modules" dodaj biblioteki:
ws2_32.lib, mswsock.lib, advapi32.lib, oraz
omniORB405_rt.lib, omniDynamic405_rt.lib, omnithread30_rt.lib

Aby móc śledzić wykonywanie programu (fragmenty omniORB), należy zamiast powyższych bibliotek dołączyć ich debbugowalne wersje:
omniORB405_rtd.lib, omniDynamic405_rtd.lib, omnithread30_rtd.lib

Dla VC++ 7 skonfiguruj projekt dla serwera (podobnie będzie dla klienta). Przed kompilacją zapoznaj się z dokumentację omniORB. Konfiguracja projektu:

a) Dodaj do projektu pliki wygenerowane przez kompilator omniidl oraz źródło serwera
(Project ->"Add existing Item"; HelloInterface.h, HelloInterface.cpp, HelloInterface_server.cpp)

b) Ustaw ścieżki dostępu do plików nagłówkowych i bibliotek:

1. Otwórz okno Project Properties
2. Wybierz C++ -> General(Additonal Include Directories) i wprowadź ścieżkę dostępu do katalogu include w zainstalowanej dystrybucji omniORB (tj. <Top-Level Directory>\include, gdzie <Top-Level Directory> to katalog główny omniORB).
3. Wybierz Linker -> General (Additional Library Directories), i wprowadź ścieżkę dostępu do katalogu bibliotek w dystrybucji omniORB (tj. <Top-Level Directory\lib\x86_win32).

c) Ustaw makra i biblioteki w projekcie:

1. W Project Properties ->Settings, wybierz zakładkę "C/C++".
2. Wybierz zakładkę "The Code Generaion":
4. Dla "Enable exception Handling" wybierz "Yes"
5. Dla "Run-Time Library" wybierz "Mulithreaded DLL".
6. Wybierz zakładkę "The Code Generaion" Tab: (pod "C++" w "project properties") i w polu "Preprocessor Definitions" dodaj makra:

__WIN32__,__x86__,_WIN32_WINNT=0x0400

Jeśli jest to NT 4.0, Windows 2000, XP, Windows 2003 Server, dodaj makra:

__NT__ , __OSVERSION__=4

7. Rozwiń zakładkę "Linker"
8. Wybierz zakładkę "Input"
9. Dodaj biblioteki w "Additional Dependencies"

ws2_32.lib, mswsock.lib, advapi32.lib, omniORB405_rt.lib, omniDynamic405_rt.lib, omnithread30_rt.lib

Aby móc śledzić wykonywanie programu (fragmenty omniORB), należy zamiast powyższych bibliotek dołączyć ich debbugowalne wersje:

omniORB405_rtd.lib, omniDynamic405_rtd.lib, omnithread30_rtd.lib

Dla kompilacji ze statycznymi wersjami bibliotek omniORB i omnithread, makra powinny wyglądać nieco inaczej:

_WINSTATIC

zaś dołączane biblioteki zamiast:

omniORB405_rt.lib, omniDynamic405_rt.lib omnithread30_rt.lib

powinno być:

omniORB4.lib, omniDynamic4.lib i omnithread.lib

Uruchom aplikacje

1. Upewnij się, że ścieżka systemowa PATH zawiera ścieżkę dostępu do katalogu bibliotek DLL z dystrybucji omniORB:
   C:\omniORB-4.0.5\bin\x86_win32\
2. Podobnie zapewnij dostęp do wykonywalnych plików z j2sdk.
3. Wystartuj serwis nazw:
   start e:\omniORB-4.0.5\bin\x86_win32\omniNames.exe -logdir . -start 3020 -ORBsupportBootstrapAgent 1

   Uwaga: jeśli przy pierwszym uruchomieniu w bieżącym katalogu znajdują się logi wygenerowane wcześniej przez omninames, należy je usunąć.
4. Uruchom jednego serwera:
   a) dla Javy:
      start java -classpath . -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -Djava.naming.provider.url=iiop://localhost:3020 HelloServer
   b) dla omniORB:
      start omniHelloInterface_server.exe -ORBInitRef nameService=corbaloc:iiop:localhost:3020/NameService
5. Uruchom klienta:
   a) dla Javy:
      start java -classpath . -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -Djava.naming.provider.url=iiop://localhost:3020 HelloClient
   b) dla omniORB:
      start omniHelloInterface_client.exe -ORBInitRef nameService=corbaloc:iiop:localhost:3020/NameService