Objetos distribuídos

Java IDL

Roteiro

- Java IDL
- Definindo a interface IDL
- Compilando a interface IDL
- Criando o servidor
- Criando o cliente
- Rodando a aplicação
- Rodando a aplicação em duas máquinas

Java IDL

- Implementação da arquitetura CORBA
- Aderente às especificações OMG
  - CORBA 2.3.1
  - Mapeamento IDL para Java
- Ferramentas
  - Compilador IDL para Java
  - Compilador Java para IDL
- Página

http://java.sun.com/j2se/1.5.0/docs/guide/idl/

Definindo a interface IDL

- Crie o arquivo Hello.idl

```
module HelloApp {
   interface Hello {
      string sayHello();
      oneway void shutdown();
   };
};
```

Compilando a interface IDL

- Mapeando Hello.idl para Java

1. Verifique se o diretorio jdk/bin está no seu PATH
2. Abra o Prompt do DOS
3. Vá para o diretório onde está o arquivo Hello.idl
4. Compile o arquivo IDL

idlj -fall Hello.idl

Compilando a interface IDL

- Foi criado o diretório HelloApp contendo 6 arquivos
  - Hello.java
    - Esta interface contém a versão Java da interface IDL
  - HelloOperations.java
    - Esta interface contém os métodos da interface Java
  - _HelloStub.java
  - HelloPOA.java
  - HelloHelper.java
    - Esta classe provê funções auxiliares
    - Método narrow() faz o cast de referência a objeto CORBA para o tipo certo
    - Responsável por ler e escrever dados em streams CORBA
  - HelloHolder.java
    - Usada quando os parâmetros são out ou inout

Criando o servidor

```
// Stub e skeleton
import HelloApp.*;
// Serviço de nomes
import org.omg.CosNaming.*;
// Exceções do serviço de nomes
import org.omg.CosNaming.NamingContextPackage.*;
// Necessário em aplicações CORBA
import org.omg.CORBA.*;
// Portable Server Inheritance Model
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
// Para iniciar o ORB
import java.util.Properties;
```


Criando o servidor

```
class HelloImpl extends HelloPOA {
   private ORB orb;
   public void setORB(ORB orb_val) {
      orb = orb_val;
   }
   public String sayHello() {
      return "\nHello world !!\n";
   }
   method public void shutdown() {
      orb.shutdown(false);
   }
}
```

Criando o servidor

- Declara a classe

```
public class HelloServer {
   // o método main() vai aqui
}
```

- Define o main()

```
public static void main(String args[]) {
   // o bloco try-catch vai aqui
}
```

- Trata exceções

```
try{
   // o resto do código de Hello Server vai aqui
} catch(Exception e) {
   System.err.println("ERROR: " + e);
   e.printStackTrace(System.out);
}
```


Criando o servidor

- Cria e inicia um objeto ORB

```
ORB orb = ORB.init(args, null);
```

- Obtém uma referência a RootPOA e ativa POAManager

```
POA rootpoa =
    POAHelper.narrow(orb.resolve_initial_referen
    ces("RootPOA"));
 rootpoa.the_POAManager().activate();
```

Criando o servidor

- Inicia objeto servant

```
HelloImpl helloImpl = new HelloImpl();
```

- Define ORB para chamar ORB.shutdown()

```
helloImpl.setORB(orb);
```

- Obtém referência associada ao servant

```
org.omg.CORBA.Object ref =
    rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.narrow(ref);
```


Criando o servidor

- Obtém contexto de nome inicial

```
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
NamingContextExt ncRef =
    NamingContextExtHelper.narrow(objRef);
```

- Registra o servant

```
String name = "Hello"; NameComponent path[] =
    ncRef.to_name( name );
ncRef.rebind(path, href);
```

- Aguarda requisições

```
orb.run();
```

Criando o cliente

```
// Stub e skeleton
import HelloApp.*;
// Serviço de nomes
import org.omg.CosNaming.*;
// Exceções do serviço de nomes
import org.omg.CosNaming.NamingContextPackage.*;
// Necessário em aplicações CORBA
import org.omg.CORBA.*;
```


Criando o cliente

- Declara a classe cliente

```
public class HelloClient {
  // o método main() vai aqui
}
```

- Define o método main()

```
public static void main(String args[]) {
  // o bloco try-catch vai aqui
}
```

- Trata exceções

```
try{
  // adicione o resto do código do cliente aqui
} catch(Exception e) {
  System.out.println("ERROR : " + e); e.printStackTrace(System.out);
}
```

Criando o cliente

- Cria e inicia um objeto ORB

```
ORB orb = ORB.init(args, null);
```

- Obtém contexto de nome inicial

```
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameServi
    ce");
NamingContextExt ncRef =
    NamingContextExtHelper.narrow(objRef);
```


Criando o cliente

- Resolve o nome e obtém a referência ao objeto

```
String name = "Hello";
helloImpl = HelloHelper.narrow(ncRef.resolve-
    str(name));
System.out.println("Obtained a handle on server
    object: " + helloImpl);
```

- Invoca operação sayHello()

```
System.out.println(helloImpl.sayHello());
```

- Invoca operação shutdown()

```
helloImpl.shutdown();
```

Compilando

- Para compilar o servidor, no diretório Hello faça:

javac HelloServer.java HelloApp/*.java

- E para compilar o cliente faça:

javac HelloClient.java HelloApp/*.java

Rodando a aplicação

- Inicie o orbd

start orbd -ORBInitialPort 1050 -ORBInitialHost localhost

- Inicie o servidor

start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost

- Execute a aplicação cliente

java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost

- O cliente imprime

Hello world!!

Rodando a aplicação em duas máquinas

- Na máquina do servidor
  - Inicie o orbd

start orbd -ORBInitialPort 1050 -ORBInitialHost maquina_servidor

  - Inicie o servidor

start java HelloServer -ORBInitialPort 1050

- Na máquina do cliente
  - Execute a aplicação cliente

java HelloClient -ORBInitialPort 1050 –ORBInitialHost maquina_servidor