

**Laboratório de Computação VI**

# ***JAVA IDL***

**Fabricio Aparecido Breve - 981648-9**

## O que é Java IDL?

Java IDL é uma tecnologia para objetos distribuídos, ou seja, objetos em diferentes plataformas interagindo através de uma rede. O Java IDL é similar ao RMI (Remote Method Invocation), que suporta objetos distribuídos que sejam escritos totalmente em Java. A vantagem do Java IDL é que ele permite que objetos se interajam independentemente de terem sido escritos em Java ou em alguma outra linguagem como C, C++, Cobol, etc.

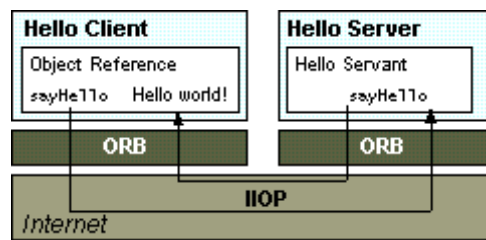
Isto é possível porque o Java IDL é baseado no CORBA (Common Object Request Brokerage Architecture), que é um padrão de indústria para o modelo de objetos distribuídos. Uma característica chave do CORBA é o IDL (Interface Definition Language). Cada linguagem que suporta CORBA tem seu próprio mapeador IDL. O Java IDL suporta este mapeamento para Java. O CORBA e os mapeadores IDL são resultado do trabalho de um consórcio de indústrias conhecido como OMG (Object Management Group). A Sun é o membro fundador da OMG, e o time do Java IDL está dando o exemplo ao definir o mapeamento de IDL para Java.

Para suportar a interação entre objetos em programas separados, o Java IDL tem o ORB (Object Request Broker). O ORB é uma biblioteca de classes Java que permite a comunicação de baixo-nível entre aplicações Java IDL e outras aplicações que suportam CORBA.

## A Arquitetura CORBA

Qualquer relação entre objetos distribuídos tem dois lados: o cliente e o servidor. O servidor tem uma interface remota e o cliente chama essa interface. Estas relações são comuns para a maioria dos padrões de objetos distribuídos, incluindo RMI e CORBA. Os termos cliente e servidor definem a interação no nível de objeto antes da interação no nível de aplicação. Qualquer aplicativo pode ser um servidor para alguns objetos e um cliente de outros objetos. Um único objeto pode ser cliente de uma interface fornecida por um objeto remoto e ao mesmo tempo implementar uma interface para ser chamada remotamente por outros aplicativos.

Abaixo temos um exemplo de como um objeto de um método distribuído é compartilhado entre um cliente CORBA e um servidor que implementa o clássico aplicativo “Hello World”:



No lado do cliente, o aplicativo inclui uma referência para o objeto remoto. O objeto referenciado tem um método que espera ser chamado remotamente. Este método na verdade está dentro do ORB, portanto a chamada ativa as capacidades de conexão do ORB, o qual passa a chamada para o servidor.

No lado do servidor, o ORB usa um código para traduzir a chamada remota em uma chamada de método do objeto local. Este código transforma a chamada e qualquer parâmetro dela para o formato específico e então chama o método. Quando o método retorna, a resposta passa por este código que transforma os resultados ou erros, e os manda de volta para os clientes através dos ORBs.

Entre os ORBs, a comunicação ocorre através de um protocolo compartilhado, o IIOP – Internet Inter-Orb Protocol, o qual é baseado no protocolo TCP/IP, e define como os ORBs do CORBA transferem informações. Da mesma forma que o CORBA e o IDL, o padrão IIOP também é definido pelo OMG.

Além dessas capacidades mais simples de objetos distribuídos, os ORBs compatíveis com CORBA podem fornecer um número de serviços opcionais definidos pelo OMG. Isto inclui serviços para procurar objetos pelo nome, manter objetos persistentes, suportar processamento de transações, habilitar comunicação, e muitas outras utilidades nos ambientes de computação distribuída de hoje. Muitos dos Java ORBs de outros fabricantes suportam algumas ou todas estas características adicionais. O ORB fornecido com o Java IDL suporta um destes serviços adicionais, a capacidade de procurar um objeto pelo nome.

## O processo de desenvolvimento para JAVA IDL

A interface para o objeto remoto é definida usando o IDL. Usa-se IDL ao invés da linguagem Java porque o compilador *idltojava* automaticamente mapeia a partir de IDL, gerando todos os arquivos fontes para Java, juntamente com a base do código para conexão com o ORB. Além disso, usando IDL é possível a outros desenvolvedores implementar clientes e servidores em qualquer outra linguagem que suporte CORBA.

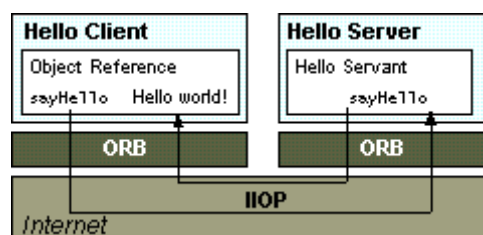
É importante lembrar que se estamos implementando um cliente para um serviço CORBA já existente, ou um servidor para um cliente já existente, obtemos as interfaces IDL com o provedor de serviços ou vendedor. Então iremos executar o compilador *idltojava* nessas interfaces e seguir os passos para criar o cliente ou o servidor.

Quando executamos o compilador *idltojava* em um arquivo de interface remota, ele gera a versão Java dessa interface, bem como os arquivos fontes para as classes que permitem que seus aplicativos se conectem ao ORB.

Uma vez que executamos o compilador *idltojava*, podemos usar os códigos que ele gera para colocar juntamente com o aplicativo servidor. Além de implementar os métodos da interface remota, os códigos do servidor incluem um mecanismo para iniciar o ORB e aguardar por um chamado do cliente remoto.

Similarmente, utilizamos os códigos gerados pelo compilador *idltojava* como base para o aplicativo cliente. Estes códigos iniciam o ORB, procuram o servidor usando o serviço de procura de objetos por nome fornecido junto com o Java IDL, obtém uma referência para este objeto remoto, e chama seu método.

## O exemplo do Hello



1. O cliente (aplicativo ou applet) chama a operação Hello do HelloServer

2. O ORB transfere a chamada para o objeto registrado com a interface IDL
3. O método `sayHello` do servidor é executado, retornando uma string Java
4. O ORB transmite essa string para o cliente
5. O cliente imprime os valores da string

## A Interface IDL

### Hello.idl

```
module HelloApp // declaração do módulo
{
    interface Hello // declaração da interface
    {
        string sayHello(); // declaração das operações
    };
};
```

Utilizando o compilador `idltojava` neste código, serão gerados os seguintes arquivos:

**HelloImplBase.java** – esta classe abstrata é a base do servidor, fornecendo as funcionalidades básicas do CORBA para o servidor. Ele implementa a interface do `Hello.java`.

**HelloStub.java** – Esta classe é a base do cliente, fornecendo funcionalidade CORBA para o cliente. Ele também implementa a interface do `Hello.java`

**Hello.java** – esta interface contém a versão Java da nossa interface IDL. Ela contém o método único `sayHello`. A interface `Hello.java` fornece funcionalidade CORBA padrão de objetos também.

**HelloHelper.java** – esta classe fornece funções auxiliares, notadamente o método `narrow`.

**HelloHolder.java** – esta classe tem uma instância pública de um membro do tipo `Hello`. Ela fornece operações para argumentos, os quais o CORBA tem mas não mapeia facilmente para a semântica do Java

## O aplicativo cliente

### HelloClient.java

```
import HelloApp.*; // o pacote contendo nossas bases
import org.omg.CosNaming.*; // serviço de nomes
import org.omg.CORBA.*; // toda aplicação CORBA precisa desta classe
```

```

public class HelloClient
{
    public static void main(String args[])
    {
        try{
            // Criando e inicializando o ORB
            ORB orb = ORB.init(args, null);

            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);

            NameComponent nc = new NameComponent("Hello", "");
            NameComponent path[] = {nc};
            Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

            // chama o objeto servidor e mostra os resultados
            String Hello = helloRef.sayHello();
            System.out.println(Hello);

        } catch (Exception e) {
            System.out.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}

```

## ***O aplicativo servidor***

### **HelloServer.java**

```

// o pacote que contém nossas bases
import HelloApp.*;

// serviço de nomes que será usado pelo servidor
import org.omg.CosNaming.*;
// o pacote contendo as exceções especiais lançadas pelo serviço de nomes
import org.omg.CosNaming.NamingContextPackage.*;
// todas as aplicações CORBA precisam desta classe
import org.omg.CORBA.*;

class HelloServant extends _HelloImplBase
{
    public String sayHello()
    {
        // retornando a string "Hello World" quando o método for chamado
        return "\nHello world !!\n";
    }
}

public class HelloServer {

    public static void main(String args[])
    {
        try{

```

```
// criando e inicializando o ORB
ORB orb = ORB.init(args, null);

// criando o servidor e o registrando com o ORB
HelloServant helloRef = new HelloServant();
orb.connect(helloRef);

org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
NamingContext ncRef = NamingContextHelper.narrow(objRef);

// ligando a referência ao objeto no serviço de nomes
NameComponent nc = new NameComponent("Hello", "");
NameComponent path[] = {nc};
ncRef.rebind(path, helloRef);

// aguardar requisições feitas pelo cliente
java.lang.Object sync = new java.lang.Object();
synchronized (sync) {
    sync.wait();
}

} catch (Exception e) {
    System.err.println("ERROR: " + e);
    e.printStackTrace(System.out);
}

}
}
```