Mapeamento de IDL para Java

Tecgraf PUC-Rio

fevereiro de 2011

Mapeamento de interface

- Uma `interface` IDL é mapeada para:
  - uma interface java de **assinatura**
  - uma interface java de **operações**
- A interface de assinatura possui o mesmo nome da interface IDL, estende outras interfaces CORBA e é usada como o tipo referenciado em outras interfaces.
- A interface de operações possui o nome da interface IDL junto com o sufixo *Operations* e é uma interface "java pura", ou seja, possui apenas as declarações das operações e atributos definidos na interface IDL
- As classes Helper e Holder são geradas

Mapeamento de interface

- Atributos definidos na interface são mapeados para um par de métodos *get* e *set*.
- Atributos readonly são mapeados apenas para métodos *get*.
- Parâmetros *in* são mapeados para parâmetros conforme o tipo correspondente.
- Parâmetros *out* e *inout* são mapeados para parâmetros do tipo *Holder* correspondente.

Mapeamento de interface

```
// IDL
module exemplo {
   struct Pessoa {
     string nome;
     long idade;
   };
   interface CadastroPessoa {
      attribute string empresa;
      void incluiPessoa (in Pessoa p);
      void alteraPessoa (inout Pessoa p);
   };
};
```

```
// Java
public interface CadastroPessoaOperations
{
        /* constants */
        /* operations  */
        java.lang.String empresa();
        void empresa(java.lang.String arg);
        void incluiPessoa(exemplo.Pessoa p);
        void alteraPessoa(exemplo.PessoaHolder p);
}
```

Mapeamento dos tipos primitivos

Tipo IDL	Tipo Java
boolean	boolean
char	char
wchar	char
octet	byte
string	java.lang.String
wstring	java.lang.String
short	short
unsigned short	short
long	int
unsigned long	int
long long	long
unsigned long long	long
float	float
double	double
fixed	java.math.BigDecimal

Mapeamento de struct

- Um `struct` IDL é mapeado para uma *final class* que possui uma variável de instância para cada campo
- A classe possui o mesmo nome do `struct` IDL
- A classe possui dois construtores:
  - um que recebe os valores iniciais para os campos
  - um construtor vazio
- As classes Helper e Holder são geradas

Mapeamento de struct

```
// IDL
struct StructType {
  long field1;
  string field2;
};
```

```
// Java
final public class StructType implements
org.omg.CORBA.portable.IDLEntity {
  // instance variables
  public int field1;
  public String field2 = "";
  // constructors
  public StructType() {}
  public StructType(int f1, String f2){...}
}
```

Mapeamento de sequence

- Um `sequence` IDL é mapeado para um array Java uni-dimensional com o mesmo nome.
- No caso de um `sequence` com tamanho máximo, a exceção **`CORBA::MARSHAL`** é lançada se o valor do array for maior do que o estabelecido
- As classes Helper e Holder são geradas

Mapeamento de sequence

```
// IDL
typedef sequence< long > UnboundedData;
typedef sequence< long, 42 > BoundedData;
```

```
// Java
final public class UnboundedDataHolder implements
org.omg.CORBA.portable.Streamable {
  public int[] value;
  public UnboundedDataHolder() {};
  public UnboundedDataHolder(int[] initial) {...};
  ...
}

final public class BoundedDataHolder implements
org.omg.CORBA.portable.Streamable {
  public int[] value;
  public BoundedDataHolder() {};
  public BoundedDataHolder(int[] initial) {...}
  ...
}
```

Mapeamento de array

- Um `array` IDL é mapeado para um array Java multi-dimensional com o mesmo nome.
- A exceção **`CORBA::MARSHAL`** é lançada se o valor do array for maior do que o estabelecido
- As classes Helper e Holder são geradas

Mapeamento de array

```
// IDL
typedef long larray[10][5];
```

```
// Java
public final class larrayHolder
        implements org.omg.CORBA.portable.Streamable
{
        public int[][] value;
        ...
}
```

Mapeamento de typedef

- Java não possui uma construção para typedef.
- Classes de *Helper* são geradas para todos os tipos definidos com a IDL `typedef`.

Mapeamento de enum

- O `enum` IDL é mapeado para uma classe com o mesmo nome.
- O tipo `int` é usado para representar cada valor da enumeração
- As classes Helper e Holder são geradas

Mapeamento de enum

```
// IDL
enum TrafficLight {
    red,
    yellow,
    green
};
```

```
// Java
public class TrafficLight implements
org.omg.CORBA.portable.IDLEntity {
  final public static int _red = 0;
  final public static int _yellow= 1;
  final public static int _green = 2;

  final public static TrafficLight red =
     new TrafficLight(_red);
  final public static TrafficLight yellow=
     new TrafficLight(_yellow);
  final public static TrafficLight green=
     new TrafficLight(_green);

public static TrafficLight from_int(int
value);
public int value() {....}
…
}
```

Mapeamento de union

- O `union` IDL é mapeado para uma classe com o mesmo nome, que possui os métodos:
  - de acesso ao valor determinante
  - de acesso e modificação para cada possível valor das opções definidas no tipo e para o valor default.
- As classes Helper e Holder são geradas

Mapeamento de valuetype

- O `valuetype` IDL é mapeado para uma classe abstrata com o mesmo nome.
  - É responsabilidade do desenvolvedor criar a classe que implenta essa classe abstrata, garantido a correta recomposição do estado do objeto no processo de *unmarshal*
- As classes Helper e Holder são geradas

Mapeamento de valuetype

```
// IDL
valuetype Node {
  public long id;
  public Node next;
};
```

```
// Java
public abstract class Node
        implements org.omg.CORBA.portable.StreamableValue {
   ...
   public int id;
   public exemplo.Node next;
   public void _write (org.omg.CORBA.portable.OutputStream os) {
     os.write_long(id);
     ((org.omg.CORBA_2_3.portable.OutputStream)os).write_value (next);
   }
   public void _read (final org.omg.CORBA.portable.InputStream os) {
     id=os.read_long();
     next=(exemplo.Node)((org.omg.CORBA_2_3.portable.InputStream)
      os).read_value ("IDL:exemplo/Node:1.0");
   }
}
```

Mapeamento de valuetype

- O desenvolvedor deve criar a classe que implementa a classe abstrata gerada a partir do `valuetype`.

```
public class NodeImpl extends Node
{
    public NodeImpl()
    {
    }
    public NodeImpl (int id)
    {
        this.id = id;
    }
    public String toString()
    {
        return "#" + Integer.toString (id) + "#";
    }
}
```

ValueFactory

- Quando uma instância de um `valuetype` chega no servidor, a classe que implementa esse objeto precisa ser encontrada para que o objeto seja reconstruído.
- O valuetype deve ter um objeto "fábrica", do tipo **ValueFactory** que é usado para instanciar a classe que implementa o `valuetype`.
- É responsabilidade do desenvolvedor implementar as classes `ValueFactory e` registrá-la no ORB

Implementação de ValueFactory

```
import java.io.Serializable;

import org.omg.CORBA.portable.ValueFactory;
import org.omg.CORBA_2_3.portable.InputStream;

/**
 * Fábrica necessária para o unmarshalling do objeto
 * que implementa o Node.
 *
 */
public class NodeFactory implements ValueFactory {

  public Serializable read_value(InputStream is) {
    return is.read_value(new NodeImpl());
  }
}
```

Registro de ValueFactory no ORB 2.3

```
orb.register_value_factory(
  NodeHelper.id(),
  new NodeFactory());
```

- Note que apenas a classe org.omg.CORBA_2_3.ORB possui os métodos que permitem registrar e remover as fábricas do ORB

Mapeamento de ANY

- tipo Any usado na IDL é mapeado para a classe Java org.omg.CORBA.Any.
- Essa classe possui todos os métodos necessários para inserir e extrair valores dos tipos primitivos.
- Se o método de extração usado for para um tipo diferente daquele guardado no Any, uma exceção CORBA::BAD_OPERATION é lançada.

Módulos

- Um módulo IDL é mapeado para um pacote Java com o mesmo nome.

```
// IDL
module tecgraf {

module openbus {

module DRMAA {
...
}
```

```
// pacote Java

tecgraf.openbus.DRMAA
```

Conflitos de nomes

- Em geral, os nomes usados na IDL são mapeados diretamente para os mesmos nomes em Java.
- Conflitos de nomes no mapeamento são resolvidos usando um prefixo _ no nome em Java.

Nomes reservados

- As classes <type>Helper and <type>Holder, onde <type> é um nome de um tipo na IDL.
- As classes <interface>Operations, <interface>POA, and <interface>POATie, onde <interface> é um nome de um tipo de interface na IDL.
- As palavras-chaves na linguagem Java.
- Métodos que colidem com os da classe java.lang.Object:
  - clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString e wait