Java Bindings

Available in Babel 0.9.4

Jim Leek

Center for Applied Scientific Computing

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

UCRL-PRES-205496
Table of Contents

• The Java Language Bindings
• How to generate Java Bindings
• An Example of using Java with Babel
• Problems for the Programmer
• Challenges in Writing the Bindings
SIDL integration into Java is nearly seamless

- SIDL and Java have a lot in common, and the JNI allows us to call native code just like Java code.
  - SIDL packages, classes, interfaces, and methods are called just like standard Java
    - Ex: package.Class.method();
  - No need to worry about reference counting.
  - Exceptions are caught and thrown, same as Java
  - Enums are final static ints in their own class
    - Ex: int state = package.enum.name;
Some mappings aren’t perfect
(Holder Classes)

- Java does not support pass by reference, so we have a public static inner class named Holder in each type for use as out/inout arguments
  - `sidl.Integer.Holder inout = new sidl.Integer.Holder(3);`  
  - `obj.passinout(inout);`
  - `int x = inout.get();`

- Holder classes are available for ALL types, including basic types, user defined types, and arrays.
Some mappings aren’t perfect  
(Wrapper Classes)  

• Java interfaces and abstract classes cannot hold an IOR pointer. We created another static inner class for abstract types named Wrapper.
  – Allows Babel to pass abstract types as method arguments and return them.
  – Allows Babel casting on abstract types.
  – Allows throwing and catching Exception Interfaces.
Some mappings aren’t perfect (Babel casting)

- When Java casting is insufficient, use a Babel cast.
  - `bar x = (bar) bar._cast(fooArray.get(2,3));`

- When is Babel cast necessary?
  - Whenever a sub class is taken out of an array of or passed as a super class/interface.

- Why is a Babel cast necessary?
  - When objects are passed by Babel or an object is retrieved from a SIDL array, a new object is created and the IOR placed inside. Java doesn’t know the IOR type, so a Babel cast is necessary to downcast it.
Every Type has an Array

- Arrays are static inner classes, every type has them. (Including basic types)
  - Array(int dim, int[] lower, int[] upper, boolean isRow)
  - foo.Bar.Array objArray =
    new foo.Bar.Array(5,0,0,0,0,0,0,0,0,true);
  - sidl.Integer.Array intArray =
    new sidl.Integer.Array(5,0,0,0,0,0,0,0,0,0,0,0,true);
- Every Array class also has numbered array subclasses that make things easier.
  - foo.Bar.Array1 arry1 = new foo.Bar.Array1(5,true);
Table of Contents

- The Java Language Bindings
- How to generate Java Bindings
- An Example of using Java with Babel
- Problems for the Programmer
- Challenges in Writing the Bindings
Generating Java bindings

- **Client side:**
  - `%babel --client=Java file.sidl`
  - `%babel --cJava file.sidl`

- **Server side:**
  - `%babel --server=Java file.sidl`
  - `%babel --sJava file.sidl`

Stub and Skeleton files are generated in the current directory, named _jniStub and _jniSkel respectively. Java files go in a directory hierarchy that duplicates the package hierarchy.
Table of Contents

• The Java Language Bindings
• How to generate Java Bindings
• An Example of using Java with Babel
• Problems for the Programmer
• Challenges in Writing the Bindings
main(String args[]) {
    Employee.Array1 empArray = new Employee.Array1(3, true);
    String[] name = {"John Smith", "Jackie Choi", Barney Rubble"};
    int[] salary = {"5232", "2134", "8792"};
    for(int i = 0; i < 3; ++i) {
        Employee emp = new Employee();
        emp.init(name[i], salary[i]);
        empArray.set(i, emp)  }
    int maxSalary, index;
    for(int i = 0; i < 3; ++i) {
        if(empArray.get(i).getSalary() > maxSalary) {
            maxSalary = empArray.get(i).getSalary; index = I; } }
    System.out.println(empArray.get(index).getName() + " has a big salary");
public class Employee_Impl extends Employee {
    // DO-NOT-DELETE splicer.begin(objarg.Employee._data)
    private String d_name = "";
    private int d_salary = 0;
    // DO-NOT-DELETE splicer.end(objarg.Employee._data)
    public void init_Impl (/* in */ java.lang.String name, /* in */ int salary) {
        // DO-NOT-DELETE splicer.begin(objarg.Employee.init)
        d_name = name;
        d_salary = salary;
        return;
        // DO-NOT-DELETE splicer.end(objarg.Employee.init)
    }
    public java.lang.String getName_Impl () {
        // DO-NOT-DELETE splicer.begin(objarg.Employee.getName)
        return d_name;
        // DO-NOT-DELETE splicer.end(objarg.Employee.getName)
    }
}
Table of Contents

• The Java Language Bindings
• How to generate Java Bindings
• An Example of using Java with Babel
• Problems for the Programmer
• Challenges in Writing the Bindings
Interface Wrappers

• Wrapper Classes are used when:
  – An object is retrieved from an interface array
  – An interface is passed to, or returned from, a Babel method
  – An interface is used as an exception.

• When does the Babel user see them?
  – When an interface is used as an exception.
  – Sometimes necessary for using BaseClass methods

• Why?
  – Java understands interfaces being returned from a method, but Exceptions must be a class.
Interface Exception Example

- **Client Side**
  
  ```java
  try{
      obj.thrw();
  } catch(example.iException.Wrapper) {/*do nothing*/}
  ```

- **Server Side**
  
  ```java
  public int thrw_Impl () throws example.iException.Wrapper {
      // DO-NOT-DELETE splicer.begin(ExceptionTest.Fib.getFib)
      iException.Wrapper ex = new iException.Wrapper(); ex.setNote("You called thrw!");
      throw ex;
      // DO-NOT-DELETE splicer.end(ExceptionTest.Fib.getFib)
  }
  ```
Table of Contents

- The Java Language Bindings
- How to generate Java Bindings
- An Example of using Java with Babel
- Problems for the Programmer
- Challenges in Writing the Bindings
Babel arrays in Java are a little ‘Different.’

- **Array Hierarchy**
  - Each type has a basic Array class, and 7 subclasses, one for each dimension.
    - Special conversion function `_dcast()`
    - Long series of minor type changes whenever working through Array hierarchy.

- **Object Arrays**
  - All object arrays actually hold a `sidl.BaseClass` array to hold the data.
Debugging

- Debugging the JNI is nightmare.
  - No tools.
    - No debugger can do naturally do both JAVA and native code.
  - Very little documentation on calling Java from C.
  - Java Garbage Collection causes unpredictable results.
Reference Counting

• Reference counting is taken care of by Java and Babel.
  – User has no choice about getting rid of data, must keep it all until Java lets go.
  – Casts must addRef(). (Unlike every other Babelized language).
  – Must be careful to always have java deleteRef when collecting a Java object.

• Of course, all of this caused plenty of trouble and was very difficult to debug.
Unexpected Exceptions
(Server Side)

• What do you do with a Java runtime Exception?
  – Not a lot you can do.
    • Can’t transmit it
    • Can’t convert it
  – Just print the message and a stack trace to Standard Error, and keep going..

• What about unexpected SIDL Exceptions?
  – Shouldn’t ever happen. Requires changing code outside spliced blocks in the _Impl file.
    • All you can do is print a message and keep going…

• This problem appears in Python and C++ too