Р11-2002-82

В. В. Галактионов

ПОТОКИ ДАННЫХ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ С **CORBA**-АРХИТЕКТУРОЙ

Предисловие

Как уже отмечалось в публикации [1], компьютерные **распределенные системы** представляют собой программные комплексы, компоненты которых (процессы) выполняются в сети на различных платформах; они связаны отношениями "клиент-сервер" и, применяя при этом различного уровня технологии - от непосредственного использования сокетов TCP/IP до технологий с высоким уровнем абстракции, таких, как RMI или CORBA, создают видимость единого целого вычислительного процесса. Основными видами взаимодействия таких процессов являются:

- синхронизация их выполнения (операции типа START/STOP и SUSPEND/RESUME);
- обмен данными.

Тип данных

Типы данных в распределенных системах с архитектурой CORBA определены стандартом языка **IDL** (Interface Definition Language) [2]. При выборе конкретного языка реализации CORBA-приложений должна выполняться процедура отображения (mapping) IDL-данных в представление этого языка. В Приложении 1 приведена таблица соответствия примитивных типов (integer,string и др.) данных IDL и языков Java и C++. Кроме того, средства IDL (например, **typedef**) позволяют пользователю создавать более сложные конструкции, например: массивы строк (тип **string**) или массивы структур (тип **struct**).

CORBA-объекты

Фундаментальным понятием для рассматриваемых распределенных систем является CORBA-объект – программная структура, выполняющаяся в стандартных случаях как серверное приложение. CORBA-объект содержит набор функций, называемых методами объекта, которые выполняются при обращении к ним клиентских программ. При этом, с точки зрения объектно-ориентированных языков программирования, таких как Java и C++, CORBA-объект является стандартной конструкцией, полностью совместимой и с понятием собственного объекта. Для тех и других объектов должны быть сначала определены переменные **object reference** (в русском переводе - *переменные экземпляра* объекта) и затем выполняться обращения к *методам* объекта. Разница заключается лишь в конструировании этих переменных, в таблице названных как **objRef**.

Пример обращения к локальным и удаленным объектам:

Локальные объекты	**CORBA-объекты (VisiBroker)**
Hello **objRef** = new Hello(); String s = **objRef**.sayHello("Alex") System. out.println(s);	ORB orb = ORB.init(); Hello **objRef** = HelloHelper.bind(orb, "HelloServer"); String s = **objRef**.sayHello("Alex"); System.out.println(s);

Обмен данными клиента с серверным CORBA-объектом выполняется через параметры методов (функций) объекта, например:

```
String res = obgRef1.funct1(x, y, z);
int    i   = objRef2.funct2(a, b, c);
float  f   = objRef3.fu();
objRef3.f(q,w);
```

Типы параметров методов CORBA-объектов

Данные прграммы-клиента передаются методам CORBA-объекта как параметры типа **IN** или **INOUT.** Результаты возвращаются как значения функций или в параметрах типа **INOUT** и **OUT.** Применение параметров типа INOUT и OUT создает дополнительную сложность в реализации обменов данными, поскольку программирование их является непростой задачей даже при передаче сравнительно простых типов данных. В CORBA предусмотрен особый механизм обработки таких параметров с возвращаемыми значениями – вводятся специальные классы и объекты типа **Holder** с соответствующими методами доступа.

Еще более сложной задачей является передача **массивов данных** в параметрах этого типа, но практика показывает, что этот режим является наиболее востребованным в распределенных системах. Типичный пример – стандартная задача с несложным и коротким запросом клиентской программы на поиск информации в базе данных сервера. При этом возвращаемые потоки данных могут быть весьма значительными.

Особая роль в распределенных системах отводится для обмена **двоичными** потоками данных, например при съеме и обработке экспериментальных данных. При применении Java-технологии в CORBA этот механизм может успешно использоваться также для передачи Java-объектов в сети. При этом используется такое замечательное свойство Java-объектов, как **сериализация**. При выполнении этой процедуры объекты, с различной степенью сложности, превращаются в массивы байтов, и дальнейшая манипуляция с ними является лишь вопросом техники программирования. Разумеется, неотъемлемой частью этой проблемы является восстановление объектов (десериализация) из байтового потока.

Очень важной особенностью взаимодействия процессов в CORBA является так называемый режим **callback.** При этом, в отличие от стандартного применения, формируются два CORBA-объекта, соответственно для клиентской и серверной программы. В этом случае CORBA-объект сервера может брать на себя инициативу передачи данных объекту программы клиента. Интерфейсы обоих объектов описываются стандартным образом в IDL-модуле. Различие формирования таких объектов заключается в разных способах создания *переменных экземпляров,* они же переменные типа **object reference** (см. выше). Переменная экземпляра серверного CORBA-объекта формируется обычным для CORBA способом, с использованием механизма локализации объектов в сети, а переменная экземпляра объекта клиента сообщается серверу клиентской программой в качестве параметра в одном из предварительных обращений. Хорошим примером демонстрации возможного применения режима **callback** является периодическая рассылка по подписке (без повторных запросов) изменяющейся информации (например, курса акций) зарегистрированным клиентам. Важность применения такого режима очевидна, тем не менее, во многих руководствах по CORBA-технологиям такая возможность даже не упоминается. Этому есть ряд объяснений, напрмер, отсутствие поддержки такого режима при клиентских приложениях в виде Java-апплетов (из-за проблемы безопасности клиентских операционных систем Java-апплеты имеют ряд серьезных ограничений). Нельзя также недооценивать сложность понимания программирования задачи коммуникации двух CORBA-объектов, задания и реализации их интерфейсов.

Надо отметить, что программирование описанных выше задач является достаточно сложным и усугубляется практическим отсутствием необходимых методологий или рекомендаций в имеющейся (или доступной автору) документации. Причина проста: в силу сложности CORBA-технологии основное внимание в литературе уделяется

разъяснениям CORBA-концепции распределенных систем, методам разработки архитектуры клиент-серверных приложений, проблемам регистрации, локализации и описания интерфейсов распределенных объектов и т.д. И все это демонстрируется на достаточно простом примере, приведенном в Приложении 2 и известном как **HelloWorld**

Надо сделать ряд предварительных замечаний для понимания последующего изложения:

- приводятся примеры использования CORBA-пакета VisiBroker компании Inprise;
- поскольку обмен примитивными типами данных (типа integer, float, string) не представляет особых затруднений, в последующих описаниях не приводятся примеры с подобными операциями;
- в примерах приводятся только фрагменты программ, существенные для описываемых случаев, поскольку обрамляющие их коды являются стандартными и одинаковыми для всех примеров применения пакета **VisiBroker**. Надо отметить также, что приведенные примеры во многом совместимы с другими CORBA-разработками, например с пакетом **Java IDL** компании Sun. Примеры полного текста образца программ и технологии разработки и их запуска приведены в работе [1] и частично в Приложении 2;
- каждый пример состоит из трех частей: образца модуля описания интерфейса (файл с расширением имен **.idl**), фрагмента серверного CORBA-объекта и фрагмента клиентской программы с подготовкой и обработкой параметров;
- все примеры сопровождаются необходимыми пояснениями как общего характера, так и комментариями к отдельным операторам программы.

1. Передача массивов данных

В примере представлен образец обмена массивами данных (целых чисел и строк) между клиентской программой и серверным CORBA-объектом. Демонстрируются способы синтаксического задания массивов данных в интерфейсном IDL-модуле и применения классов типа **Holder**, генерируемых IDL-компилятором, для хранения сложных структур, каковыми являются массивы.

Файл описания интерфейса array.idl

В описании интерфейса применен опрератор **typedef** для создания нового типа данных – массивов для целых чисел и строковых данных. Приведено также описание четырех методов объекта (getInt(), getPutInt(), getString(), getPutString()), параметрами которых и возвращаемыми значениями являются массивы.

```
module Arrays {
  interface ArrayData  {
    typedef sequence<long> Idata;
    typedef sequence<string> Sdata;
    Idata getInt();
    Idata getPutInt(in Idata rd);
    Sdata getString ();
    Sdata getPutString(in Sdata rd);
  };
};
```

Фрагмент программы клиента

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
ArrayData ARef = ArrayDataHelper.bind(orb, "ArrayServer");
//---  подготовка рабочих массивов данных-----
int[]      intArr = new int[20];
String[] strArr = new String[20];

for(int j=0; j < 20; j++) {
          intArr[j] = j;
          strArr[j] = "String " + j;
}
//---- создание объектов типа Holder для хранения сложных типов
//       данных  (массивов)
IdataHolder intDat = new IdataHolder(intArr);
SdataHolder strDat = new SdataHolder(strArr);

//----- обращение к методу getInt() для получения массива целых чисел
//------ и распечатка результата
 int len = 0;
 int[]   outArr  =  ARef.getInt();
 len = outArr.length;
 for(int i = 0; i < len; i++)
      System.out.print(" " + outArr[i]);

 //---- обращение к методу getPutInt() для передачи массива чисел,
 //      получение и распечатка  результата
  int[]   outArr2 =  ARef.getPutInt(intDat.value);
  len = outArr2.length;
  for(int i = 0; i < len; i++)
      System.out.print(" " + outArr2[i]);

 //----- обращение к методу getString() для приема массива
 //       строковых данных и их распечатка
 String[] outStr  =  ARef.getString();
 len = outStr.length;
 for(int i = 0; i < len; i++)
      System.out.print(" " + outStr[i]);

 //----- обращение к методу getPutString() для передачи и приема
 //       массива строковых данных и их распечатка
 String[] outStr2 =  ARef.getPutString(strDat.value);
  len = outStr2.length;
  for(int i = 0; i < len; i++)
      System.out.print(" " + outStr2[i]);
```

Файл CORBA-объекта с четырьмя методами

```
class ArrayServant extends _ArrayDataImplBase {
```

```
    public ArrayServant(String sn) {
      super(sn);
    }
    public  int[] getPutInt(int[] rd){
     //---- создание массива целых чисел для возврата результата
     int[] intarr = new int[rd.length];

     //---- проверка принятого массива и заполнение
     //      массива результата
     for(int i=0; i < 20; i++) {
       intarr[i] = rd[i]*i;
       System.out.print(" " + rd[i]);
      }
     return intarr;
    }
    //---------------------------------------
    public  int[] getInt(){
     int[] intarr = new int[30];
     for(int i=0; i < 30; i++) {
       intarr[i] = i;
     }
     return intarr;
    }
    public  String[] getPutString(String[] rd){

      //---- создание массива строк для возврата результата
     String[] strArray = new String[rd.length];

     //---- проверка принятого массива и заполнение
     //      массива результата
     for(int i=0; i < 20; i++) {
       strArray[i] = "Result " + rd[i];
       System.out.print(" " + rd[i]);
     }
     return strArray;
    }
    //---------------------------------------
    public  String[] getString(){
     String[] strArray = new String[20];
     for(int i=0; i < 20; i++) {
       strArray[i] = "Result " + i;
     }
     return strArray;
    }
}
```

2. Передача структур данных

Пример содержит образец обмена между клиентской и серверной программами простейшей структурной единицей данных, состоящей из двух элементов – целого числа и текстовой строки.

Файл описания интерфейса st.idl

```
module STRUCTUR {
   interface stru   {
      struct st {          //------ описание структуры на языке IDL
        long ID;           //------ целое число
        string str;        //------ текстовая строка
      };
      //---- метод CORBA-объекта. Содержит два параметра типа IN :
      // ----                    строку текста и структуру
      //---  возвращаемый результат: новая структура
      st getStruct(in string s, in st struc);
   };
};
```

Файл с CORBA-объектом stServant.java

```
class stServant extends _struImplBase {
   //------  конструктор объекта
   public stServant(String str){
      super(str);
   }
   //----- метод объекта
   public st getStruct(String name, st struct) {
     System.out.println("Recieve struct: " + struct.ID + "," +
                    struct.str);
     st stu = new st(20, name);
     return stu;
   }
}
```

Фрагмент программы клиента.

```
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
   strRef = struHelper.bind(orb, "Struct");
  //------ создание структуры
   st struct = new st(10, "TEST for structure");

   //------ обращение к методу getStruct CORBA-объекта с двумя
   //       параметрами
   st res = strRef.getStruct("Name", struct);

   //------ распечатка полученного результата
   System.out.println("Res: " + res.ID + "," + res.str);
```

3. Передача массивов структур данных

Пример содержит образец обмена между клиентской программой и серверной программой (CORBA-объектом) массивами, элементами которых являются структуры данных из предыдущего примера. В этом примере используются классы типа **Holder**,

генерируемые IDL-компилятором, для хранения сложных структур данных (не примитивных, типа int, float и др).

Файл IDL-описания интерфейса

```
module V {
  interface Ve   {
    struct vectorItem {  //-- описание элемента массива
        long ID;
        string str;
    };
    //----- нового типа данных как массива структур
    typedef sequence<vectorItem> Vect;

   //------ описание двух методов putVector и getVector
   //------ оба с параметрами типа IN
    void putVector(in Vect vec);
    //----- возвращаемый результат  – новый массив структур
    Vect getVector(in string st);
  };
};
```

Фрагмент программы клиента

```
    Ve VRef = null;
    // Инициализация  ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    VRef = VeHelper.bind(orb, "VectorStruct");

    // Создание и заполнение массива структур
    vectorItem[] vit = new vectorItem[10];
    VectHolder vv = new VectHolder(vit);

     for(int j=0; j < 10; j++) {
        String st = "###" + j + j + j;
        vv.value[j] = new vectorItem(j, st);
      }
      //----  проверка (распечатка) передаваемого массива структур
       int l = vit.length;
       for(int i = 0; i < l; i++) {
           jj = vit[i].ID;
           str = vit[i].str;
           System.out.println(str + " " + jj);
       }
       //---- передача созданного массива методу putVector()
       //      CORBA-объекта
       VRef.putVector(vit);

       //---- Прием массива структур от мктода -- getVector()
       //     CORBA-объекта
       vectorItem[] vitm = VRef.getVector("Name");
```

```
//---- распечатка принятого массива структур
int len = vitm.length;
 for(jj=0; jj< len; jj++) {
     String sr = vitm[jj].str;
     k = vitm[jj].ID;
     System.out.println("ID=" + k + ", text=" + sr);
 }
```

Файл CORBA-объекта VServant.java

```
class VServant extends _VeImplBase {

   public VServant(String str){       //---- конструктор CORBA-объекта
    super(str);
   }
   //-------- первый метод putVector(). Входной параметр – массив структур
   public void putVector(vectorItem[] vit) {

    int l = vit.length;
    //---- проверка принятого массива
    for(int i = 0; i < l; i++) {
          int   jj = vit[i].ID;
          String str = vit[i].str;
          System.out.println(str + " " + jj);
   }
   }
   //----- второй метод, возвращаемый результат – массив структур
   public vectorItem[] getVector(String name) {
     Vector list = new Vector();
     String str;
     int jj;

     //----  промежуточная структура данных  (типа Vector) для результата
     for(int i=0; i < 10; i++) {
        str = "";
        for(int j=0; j<10; j++) str = str + i;
        list.addElement(str);
     }
     //--- создание массива для возврата результата
     vectorItem[] vit = new vectorItem[10];
     VectHolder vv = new VectHolder(vit);

      //------ заполнение массива
      jj = 0;
      for(Enumeration en = list.elements(); en.hasMoreElements(); jj++ ) {
            String st = "$$$" + en.nextElement().toString();
            vv.value[jj] = new vectorItem(jj, st);
      }
```

```
        //------- проверка и возврат результата -------
        int l = vit.length;
        for(int i = 0; i < l; i++) {
            jj = vit[i].ID;
            str = vit[i].str;
            System.out.println(str + " " + jj);
        }
        return vit;
    }
}
```

4. Передача результатов в параметрах типа INOUT и OUT

Параметры типа INOUT предназначены для передачи данных методу CORBA-объекта и для размещения в них же возвращаемого результата. Параметры типа OUT предназначены только для возвращаемого результата. Проблема параметров рассматриваемого типа в языке Java заключается в том, что значения в параметрах функций передаются как **by value**, т.е. в них нельзя размещать никаких данных для возврата результатов. Поэтому для реализации требований CORBA приходится использовать даже для простых переменных специальные конструкции в виде классов, которые передаются в качестве параметров типа **by reference**, т.е. в виде ссылок. Для этой цели при компиляции IDL-интерфейсов автоматически генерируются классы типа **Holder,** которые собственно и используются для передачи сложных параметров, в том числе и параметров типа INOUT и OUT. Значения данных в этих классах содержит переменная со стандартным именем **value**. Для примитивных типов данных генерируются классы этого типа со стандартными именами, в которые включены символические имена таких данных: IntHolder, ByteHolder, FloatHolder, DoubleHolder, CharHolder, StringHolder и т.д.

В примере использованы три метода с применением параметров типа INOUT и OUT:

- string getput(in string st, **out** string os, **inout** string ios). В параметре ios метод получает входное значение текстовой строки, и туда же может быть помещена другая строка в качестве возвращаемого значения.
- Следующие два метода также демонстрируют использование параметров типа INOUT и OUT, но для передачи уже массивов целых чисел

```
    void writea(inout data wrdata);
    void writeb(out data ar);
```

Файл IDL-описания интерфейса

```
module in_out {
  interface INOUT   {
  typedef sequence<long> data;
  void writea(inout data wrdata);
  void writeb(out data ar);
    string getput(in string st,  out string os,  inout string ios);
  };
};
```

Файл CORBA-объекта inoutServant.java

```
class inoutServant extends _INOUTImplBase {
   public inoutServant(String ns) {
```

```
    super(ns);
  }
  //------ метод с параметром типа INOUT ------------------------------
  public void writea(dataHolder wr){
      System.out.println("write() started!" + wr.toString());

     //-- проверка принятого значения массива и запись в этот же
     //    массив нового значения в качестве возвращаемого результата.
      for(int j=0; j < 10; j++) {
        System.out.println("get=" + wr.value[j]);
        wr.value[j] = j*j;
       }
  }
  //---------метод с параметром типа OUT -----------------
  public void writeb(dataHolder wr){

   //---- создание нового значения переменной value
   //      в классе типа Holder
   wr.value = new int[20];

   //---- запись возвращаемого результата
    for(int j=0; j < 20; j++)
         wr.value[j] = j*j;
  }

  //------ метод с параметрами os (типа  OUT) и ios (типа  INOUT)
  public String getput(String st, StringHolder os,StringHolder  ios) {
     //--- запись возвращаемого результата через переменную value
    os.value = " 111111111111 ";
    ios.value = ios.value.toUpperCase();
     return st.toUpperCase();
  }
}
```

Фрагмент программы клиента

```
   // Инициализация ORB.
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
   INOUT ioRef = INOUTHelper.bind(orb, "InOut");

   //========== метод writea(inout p) ======================
   //---- передача массива целых чисел в параметре типа INOUT и
   //      прием результата в этом же иассиве
    int intar[] = new int[10];
    for(int j=0; j < 10; j++)   intar[j] = j;

    dataHolder outArray = new dataHolder(intar);
    ioRef.writea(outArray);

    //----- проверка результата
    for(int i=0; i < 8; i++) {
       System.out.println("Output element out array [" + i +"]="
```

```
                    + outArray.value[i]);
}

//====== метод void writeb(out data ar) ==============
//-------  параметр типа OUT
dataHolder outb    = new dataHolder();
ioRef.writeb(outb);

//--------  обработка и распечатка результата
int[] res = outb.value;
int le = res.length;
for(int i=0; i < le; i++) {
    System.out.println("res=" + res[i]);
}

//=== метод getput(in string st,  out string os,  inout string ios) =====
String          st        = new String("qwertyuiopasdfghjklzxcvbnm");
StringHolder outstr    = new StringHolder();
StringHolder inoutstr  = new StringHolder("java station");
//---- передача строки st в качестве параметра типа IN
//      и строки  inoutstr в качестве параметра типа INOUT

String Hello = ioRef.getput(st,outstr, inoutstr);
//-----  прием результата:
//      строка Hello – как  возвращаемый  результат функции,
//      строковые данные outstr и inoutstr - как  результат в параметрах
//      типа   INOUT и OUT

String ou = outstr.value;
String ios = inoutstr.value;
System.out.println(Hello + ou + ios);
```

5. Передача потоков байтов. Сериализация и передача Java-объектов

Передача массивов байтов практически не отличается от передачи массивов целых чисел, рассмотренных в одном из предыдущих примеров. Разница заключается в синтаксисе описания массивов в IDL-модуле, например:

typedef sequence<octet> bytes

Важность такого обстоятельства заключается в открывшейся возможности **передачи объектов**. Под объектом понимается классическое представление программных структур, принятое в объектно-ориентированных языках программированя, т.е. инкапсулированные в одной структуре данные и программы (методы) для доступа к этим данным. В спецификациях CORBA пока не определен механизм передачи объектов в распределенных гетерогенных системах. Вся сложность этого явления заключается в провозглашенном в архитектуре CORBA принципе *интероперабельности*, т.е возможности взаимодействия программ клиента и сервера, написанных на разных языках, соблюдая однако тем не менее требование объектной ориентации. Естественно, соблюсти совместимость программных объектов при их передаче в таких случаях пока не представляется возможным. Тем не менее, вопрос о передачах объектов стоит в повестке дня направления развития CORBA-систем.

При использовании же однородных распределенных систем, например, разрабатываемых с использованием единого языка Java, передача объектов допускается. Здесь может быть учтен опыт применения системы **RMI** (Remote Method Invocation), используемой для распределенных систем на Java-платформах. В CORBA-приложениях, разрабатываемых на языке Java, может быть эффективно использовано замечательное свойство Java-объектов – возможность их **сериализации.** Это означает, что в Java предусмотрен механизм преобразования объектов в последовательность байтов и обратного их восстановления. Эта процедура во многом совпадает для различных Java-объектов, но нельзя однозначно ответить на вопрос ее применимости для всех объектов. Часто это зависит от подготовленности и квалификации Java-специалиста.

В рассматриваемом ниже примере рассматриваются в качестве наглядного пособия следующие действия:

- создание в клиентской программе графического объекта типа Frame (окна) с компонентами GUI (графического интерфейса пользователя) и программ для обработки событий;
- сериализация этого объекта;
- передача этого массива байтов программе-методу серверного CORBA-объекта;
- восстановление на сервере принятого массива в графический объект типа Frame, модификация его и запуск в работу;
- выполнение клиентской программы запроса на прием такого же объекта от серверной программы и выполнение аналогичных действий (сериализация объекта, передача его и восстановление) программами клиента и сервера. В тексте программы даны необходимые пояснения в виде стандартных комментариев. Программа создания объекта Frame не приводится в силу тривиальности его конструирования, просто указан оператор его создания:

  Frame ft = new setFrame("Original frame");

Файл IDL-описания интерфейса

```
module Serial {
  interface bStream   {
    typedef sequence<octet> bytes;
    void writeStream(in bytes  array);
    bytes readStream();
  };
};
```

Фрагмент программы клиента

```
  //---  инициализация ORB и локализация CORBA-объекта
  org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
  Serial.bStream strRef =
       Serial.bStreamHelper.bind(orb, "Stream");

  //---- создание графического объекта (окна)
  Frame ft = new setFrame("Original frame");

  //=======  сериализация и передача объекта ======
  try {
    //-----  сериализация объекта типа Frame и размещение потока
    //        байтов в массиве bb
```

```
    ByteArrayOutputStream o = new ByteArrayOutputStream();
    ObjectOutput os = new ObjectOutputStream(o);
    os.writeObject(ft);
    byte[] bb = o.toByteArray();

    //-------- передача Frame-объекта CORBA-методу .writeStream()
    strRef.writeStream(bb);
  } catch (Exception ex) { }

//========== чтение и восстановление объекта ==========
try {
  //-------- обращение к методу readStream() CORBA-объекта
  ByteArrayInputStream bs =
     new ByteArrayInputStream(strRef.readStream());
  ObjectInput is = new ObjectInputStream(bs);
  java.lang.Object obj = is.readObject();
  is.close();

  //------ пример восстановления объекта из байтового представления
  Class cla = obj.getClass();
  String className      = cla.getName();
  Class superclass      = cla.getSuperclass();
  String superClassName = superclass.getName();

  //------ проверка типа восстанавливаемого объекта и, если он
  //       имеет тип Frame, восстанавливается графический объект и
  //       формируется его изображение
   if(superClassName.equals("java.awt.Frame")) {
      Frame f = (Frame) obj;
      f.setTitle("Recieved Frame from Server object");
      f.resize(400,200);
      f.show();
   }

} catch(Exception ex) {   }
```

Файл с методами CORBA-объекта

В программе выполняются операции с графическим объектом типа Frame (сериализация и восстановление), аналогичные и симметричные клиентской программе. Поэтому комментарии к программе сокращены.

```
class streamServant extends Serial._bStreamImplBase {
  Frame f;
  java.lang.Object obj;

  //----- конструктор объекта --------
  public streamServant(String str){
     super(str);
  }
  //---------метод readStream() в качестве результата отправляет массив
  //          байтов.
```

```
    //        В данном примере передается графический объект после его
    //        сериализации
    //        Логика прмера построена на том, что этот метод возвращает
    //        клиентской программе полученный от нее ранее графический
    //        объект типа Frame
public byte[] readStream() {
          try {
         ByteArrayOutputStream ov = new ByteArrayOutputStream();
         ObjectOutput osv = new ObjectOutputStream(ov);
         f.setTitle("Send from server");
         osv.writeObject(f);
         return ov.toByteArray();

         //----- другой вариант передачи байтового массива:
         // byte[] bb = ov.toByteArray();
         // return bb;
     } catch (Exception ex) { System.out.println("Server exception");
                  return null;
       }
 }
 //=============================================================
 //------ метод writeStream() в качестве входного параметра (массива байтов)
 //        получает графический объект типа Frame, выполняет его
 //        идентификацию и восстанавливает его изображение.

 public void writeStream(byte[] vct){
     try {
         ByteArrayInputStream bs = new ByteArrayInputStream(vct);
         ObjectInput is = new ObjectInputStream(bs);
         obj = is.readObject();
         is.close();
     } catch (Exception e) {   }

    Class cla = obj.getClass();
    String className = cla.getName();
    Class superclass   = cla.getSuperclass();
    String superClassName = superclass.getName();
    if(superClassName.equals("java.awt.Frame")) {
        f = (Frame) obj;
        f.setTitle("Recieved Frame object");
        f.resize(400,200);
        f.show();
    }
}
```

5. Режим callback

Основная идея этого режима клиент-серверного взаимодействия описана в предисловии данной работы. Здесь же на примере будет показана техника его реализации.

Задание интерфейса двух CORBA-объектов

Для нижеприведенного примера в IDL-модуле задаются два интерфейса – для клиентского (**CallBack) и** серверного (**Hello**) объекта. Клиентский объект содержит один метод **callback()** с параметром, а серверный – один метод **sayHello()** с параметром, в котором серверному объекту передается *переменная экземпляра* CORBA-объекта клиента

```
module HelloApp {
  //----- интерфейс для клиентского объекта
  interface Callback {
     void callback(in string message);
  };
  interface Hello    {
     string sayHello(in Callback objRef);
  };
};
```

Фрагмент программы клиента.

```
//======   CORBA-объект клиента
class Callbackservant extends _CallbackImplBase {
   //------ метод запускается при обращении к нему из программы
   //       серверного  CORBA-объекта
   public void callback(String notification) {
      System.out.println(notification);
   }
}
//---------------------------------------------
public class HelloClient
{
   public static void main(String args[])   {
      // Инициализация  ORB и локализация серверного CORBA-объекта
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
     Hello callRef = HelloHelper.bind(orb, "callback");

      //-----   создание клиентского CORBA-объекта и
      //         его регистрация
      Callbackservant backRef = new Callbackservant();
      orb.connect(backRef);

     //-----  обращение к методу серверного объекта и передача в
     //        качестве параметра ссылки на клиентский объект
     String hello = callRef.sayHello(backRef);
    System.out.println(hello);
  }
}
```

Программа серверного CORBA-объекта

```
class HelloServant extends _HelloImplBase
{
```

```
    //------ конструктор объекта
    public HelloServant(String ns) {
        super(ns);
    }
    //--------- этот метод вызывается из клиентской программы
    //          в параметре сообщается ссылка на CORBA- объект клиента
    public String sayHello(Callback callobj) {
        //-------- обращение к методу callback() CORBA-объекта клиента
        callobj.callback("Hello from server!");
        //------- возврат результата
        return "\nHello world !!\n";
    }
}
```

Заключение

В работе приведены результаты исследований обмена различного типа данными в распределенных системах с CORBA-архитектурой с применением инструментального пакета **VisiBroker** компании Inprise. Аналогичные исследования проводились ранее для пакета **Java IDL** компании Sun. Результаты показали незначительные различия в программировании обмена данными. Все это были предварительные исследования для решения практической задачи[3, 4], стоящей перед автором: обеспечение доступа к СУБД Oracle 8i с применением технологий распределенных систем, в частности RMI и CORBA. Направление исследований определило то обстоятельство, что компания Oracle лицензировала применение VisiBroker (и других средств Java-технологий, таких, как JSP, Java Stored Procedure и EJB, Enterprise JavaBeans) в своем программном обеспечении для баз данных, в том числе для СУБД Oracle 8i, начиная с версии 1.1.6. Это и стимулировало повышенное внимание разработчиков к этим технологиям.

Кроме того, данная работа имеет и самостоятельный интерес для применяющих CORBA-технологии, компенсируя недостаточное освещение рассмотренных проблем в литературе и фирменной документации.

Приложение 1. Таблица соответствия некоторых типов данных IDL, Java и C++

IDL	Java	C++
boolean	boolean	bool
char	char	signed char
octet, wchar	byte	8 bits
short	short	short
long	int	long
float	float	float
double	double	double
string	java.lang.String	class string
struct	class	struct
union	class	union
sequence, array	array	array

Приложение 2. Стандартный пример **HelloWorld** применения пакета **VisiBroker**

Файл **Hello.idl**

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

Файл **HelloClient.java**

```
public class HelloClient {
  public static void main(String[] args) {
    // Initialize the ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    HelloApp.Hello helloRef =
                        HelloApp.HelloHelper.bind(orb, "HelloServer");
    String hello = helloRef.sayHello();
    System.out.println(hello);
  }
}
```

Файл **HelloServer.java**

```
public class HelloServer {
  public static void main(String[] args) {
    // Initialize the ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    // Initialize the BOA.
    org.omg.CORBA.BOA boa = orb.BOA_init();

    HelloApp.Hello helloRef = new HelloServant("HelloServer");

    // Export the newly created object.
    boa.obj_is_ready(helloRef);
    System.out.println("HelloServer is ready.");
    // Wait for incoming requests
    boa.impl_is_ready();
  }
}
```

Файл **HelloServant.java**

```
class HelloServant extends HelloApp._HelloImplBase {
  public HelloServant(String str){
    super(str);
  }
  public String sayHello() {
    System.out.println("HelloWorld");
   return "\nHello world !!";
  }
}
```

Литература

1. Галактионов В.В. Java-технологии в распределенных системах с CORBA-архитектурой.Сообщение ОИЯИ, Р10-2002-63, Дубна, 2002.
2. Object Management Group.http://www.omg.org/.
3. Schmeisser Nils. Zentrale Nutzerdatenbank.Handbuch. Forschungzentrum Rossendorf e.V., Abt. Kommunikation und Datenverararbeitung, Drezden, 1999.
4. Galaktionov V., Pervouchov V., Schmeisser N. Central User Database. Protocol. Forschungzentrum Rossendorf e.V., Abt. Kommunikation und Datenverararbeitung, Drezden, 1999.

Получено 18 апреля 2002 г.

Галактионов В. В.
P11-2002-82

Потоки данных в распределенных системах
с CORBA-архитектурой

Приведены результаты исследований обмена различного типа данными в распределенных системах с CORBA-архитектурой с применением инструментального пакета VisiBroker компании «Inprise».

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2002

Перевод автора

Galaktionov V. V.
P11-2002-82

Data Exchange for Distributed Systems Based on the Use
of CORBA-Architecture Oriented Middleware

The results of the investigation for data exchange in distributed systems based on the use of package VisiBroker of company «Inprise» for CORBA oriented middleware are described.

The investigation has been performed at the Laboratory of Information Technologies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2002

Редактор *М. И. Зарубина*
Макет *Н. А. Киселевой*

ЛР № 020579 от 23.06.97.
Подписано в печать 30.05.2002.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,12. Уч.-изд. л. 1,85. Тираж 320 экз. Заказ № 53325.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.