Р10-2002-63

В. В. Галактионов

JAVA-ТЕХНОЛОГИИ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ С **CORBA**-АРХИТЕКТУРОЙ

1. Предисловие

В общем случае распределенные системы представляют собой программные комплексы, компоненты которых 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.

К настоящему времени сформировались две наиболее известные системы для разработки и развертывания сложных объектно-ориентированных прикладных приложений, а именно:

- компонентная объектная модель Component Object Model (**COM**), разработанная корпорацией Microsoft,
- общая архитектура брокеров объектных запросов Common Object Request Broker Architecture (**CORBA**), которую развивает Консорциум OMG [1].

При этом надо различать область применения этих технологий [2, 3]:

- компоненты COM (ActiveX (компонентная модель COM) и MTS (Microsoft Transaction Server)) способны функционировать только под Windows NT/Windows2000;
- CORBA, в отличие от COM'а, является *концепцией*, а не ее реализацией. В CORBA изначально была заложена многоплатформенность и поддержка множества популярных объектно-ориентированных языков программирования.

Реализации CORBA многочисленны и принадлежат множеству производителей. Эти продукты поддерживают обширный диапазон аппаратных платформ, в том числе мэйнфреймы, мини-компьютеры и Unix-системы. Inprise/Corel VisiBroker и Application Server, BEA WebLogic, Iona Orbix, Oracle Application Server, IBM BOSS - все эти продукты используют те или иные возможности CORBA.

Принципиальная особенность CORBA-архитектуры - это клиент-серверные технологии, в которых обращение к удаленному объекту предоставляется клиенту посредством использования абстрактных **интерфейсов.** Интерфейс определяет набор методов, которые реализуют функции, присущие данному объекту. Клиент получает доступ к объекту только путем вызова метода, определенного в интерфейсе объекта. Сокрытие деталей реализации и позволяет в конечном итоге добиться выполнения принципа интероперабельности – *независимости* от того, где и на какой платформе они реализованы и какой язык программирования для этого использовался. В CORBA интерфейс объекта задается с помощью определенного Консорциумом OMG *языка описания интерфейсов* (Interface Definition Language, **IDL**). Различные языки программирования поддерживаются благодаря заданным *отображениям* (mapping) между описаниями типов данных на IDL в соответствующие определения на конкретном языке. Эти отображения реализуются **компилятором IDL**, который генерирует исходные коды на нужном языке. В настоящий момент поддерживается отображение в Си, Си++, SmallTalk, Ada95, Visual Basic, Cobol и Java.

Ядром архитектуры CORBA является *брокер объектных запросов* (Object Request Broker, **ORB**). Брокеры клиента и сервера совместно ответственны за взаимодействие локальных и удаленных объектов. Помимо самого вызова метода удаленного объекта, ORB отвечают за поиск реализации объекта, его подготовку к получению и обработке запроса, передачу запроса и доставку результатов клиенту. Различные брокеры запросов взаимодействуют между собой по протоколу General Inter ORB Protocol (**GIOP**). В связи с активным переносом в среду Web распределенных приложений интерес представляет

реализация протокола GIOP на базе TCP/IP — протокол Internet Inter ORB Protocol (**IIOP**).

Наиболее используемыми языками в CORBA в настоящий момент являются **Java** (вследствие прекрасного взаимодействия Java-технологий, особенно JDBC, RMI, JNDI и EJB, с CORBA), и **C++** - как очень эффективный и распространенный язык компьютерной индустрии.

Отображения CORBA-интерфейсов в Java не требуют никаких изменений от виртуальной Java-машины. Реализации компаний Iona, Sun и Visigenic предлагают службы CORBA "времени выполнения", написанные на Java. Это означает, что в браузер можно загрузить апплет Java, который сможет обращаться к серверу CORBA без предварительной установки средств поддержки CORBA.

2. Java-технологии в распределенных системах с CORBA-архитектурой

Применение языка Java в качестве инструментального средства разработок приложений для распределенных систем явилось мощным стимуляторм их развития. В предисловии уже отмечались возможности языка Java, привлекательные с точки зрения применения их в распределенных системах. Здесь можно лишь отметить еще одно немаловажное обстоятельство: при рассмотрении трехзвенной модели распределенных систем (графический интерфейс пользователя, программы промежуточного слоя (middleware) и средства доступа к базам данных) легко видеть, что средства языка Java в полной степени могут обеспечить практическую реализацию всех звеньев, что и происходит на самом деле.

Все это означает, что Java является не только популярным языком программирования CORBA-приложений, но он привносит с собой огромный набор средств, расширяющий сферу применения самих распределенных систем. Так что речь идет о применении **Java-технологии** для этих систем.

Можно отметить ряд серьезных CORBA-реализаций с применением языка Java:

- **Java IDL** – разработка компании **Sun Microsystems Inc.** (http:\\www.javasoft.com\), вошедшая в качестве стандартных Java-средств в JDK 1.2,
- **VisiBroker 3.3** – разработка компании **Inprise Corp**. (htt: //www.inprise.com/),
- **OrbixWeb 3.1** - разработка компании **Iona Technologies Inc**. (http://www.iona.com/),
- **ILU** - разработка компании **Parc Xerox** (http://www.parc.xerox.com/),
- **OmniBroker** – разработка компании **Object-Oriented Concepts.**

Надо отметить, что пакеты программ JDK 1.2 (Sun), ILU (Parc Xerox) и OmniBroker (Object-Oriented Concepts) являются свободно распространяемыми продуктами, а VisiBroker 3.3 и OrbixWeb 3.1 – коммерческими. Впрочем, эти коммерческие программы доступны для ознакомления и исследования в виде **evaluate**-версий.

Пока еще перечень вычислительных систем (платформ), для которых применимы перечисленные реализации, недостаточно широк. Это в основном, персональные ЭВМ с **Windows 95/98/NT** и компьютеры компании Sun с операционной системой **Solaris**. Но продолжающаяся экспансия Java-технологий (для UNIX Digital, Linux, Silicon Graphics, Macintosh и др.) дает основания предположить возможное значительное расширение этого круга.

2.1. Применение апплетов в CORBA-приложениях

Использование языка Java в программировании CORBA-приложений естественным образом поставило вопрос о возможности применения для этой цели апплетов, широко

используемых в WWW Java-приложениях. Апплет – это Java-приложение, выполняющееся Web-просмотрщиком (браузером). Использование развитой технологиии WWW для распределенных систем, включая CORBA-архитектуру, может значительно увеличить область их применения. Но не следует также и преувеличивать значение применения апплетов в распределенных системах. С одной стороны, применение апплетов упрощает для конечного потребителя обращения к CORBA-серверам, ведь клиентская часть распределенной системы загружается в виде HTML-файлов с удаленных Web-серверов и выполняется стандартными Web- интерпретаторами (браузерами) типа широкоизвестных Netscape Communicator или Internet Explorer. С другой стороны, специально разработанные средства безопасности для потребителя существенно ограничивают возможности апплетов. Для CORBA-приложений это сказывается в следующем:

- выполнение CORBA-приложений в виде апплетов касается лишь клиентской части (по крайней мере в текущий момент);
- апплет-приложениям, выполняющимся под управлением Web-браузеров, запрещен доступ к файловой системе машины, на которой они выполняются;
- апплетам запрещено сетевое обращение к другим машинам, кроме той, с которой они были загружены.

Эти серьезные ограничения на выполнение апплет-программ в отдельных случаях могут привести к бесполезности их применения для распределенных систем. Тем не менее, ведутся работы по смягчению этой ситуации – в частности, можно создавать **signed**-апплеты, для которых снимаются ограничения по доступу к операционной системе потребителя. Но пока из-за сложности их конструирования и внедрения маловероятно широкое их практическое применение в ближайшем будущем.

Следует также отметить еще одно обстоятельство, осложняющее применение апплетов в CORBA. Существует специальный режим взаимодействия пары клиент/сервер – **callback**, в котором инициатива запроса на обмен принадлежит программе сервера. Особый статус выполнения клиента в виде апплетов пока не позволяет выполнять этот режим.

В практике примененииия апплетов следует различать их интерпретаторы (просмотрщики). Это программа **appletviewer** (браузер из JDK) и Web-браузеры типа **Netscape** и **Internet Explorer**. Основное отличие их с точки зрения выполнения CORBA-приложений заключается в различных наборах стандартных Java-классов. Особенно это касается классов, реализующих CORBA-спецификации и протоколы. Поскольку **appletviewer** берется из JDK, здесь не возникает больших осложнений при пропуске клиент-приложений в виде апплетов. Что же касается специализированных программных пакетов для CORBA/Java-приложений, то технология применения апплетов реализована по-разному. Более подробно эта проблема будет описана в обзоре таких разработок.

2.2. Схема реализации CORBA-архитектуры

Вычислительные системы, взаимодействующие в сетевой среде, получают статус **распределенных** систем при выполнении необходимых условий:

- определении протоколов взаимодействия двух любых машин и/или процессов как **клиент/серверной** пары,
- наличии средства **"directory services"** для локализации объектов в сетевой среде,

- наличии средства **"transaction monitor"** для наблюдения и контроля функционирования распределенной среды.

В различных программных реализациях CORBA-архитектуры вышеописанные требования находят свое специфическое разрешение и удовлетворяются своим набором функций.

Независимо от языка программирования CORBA-приложений существует единая схема подготовки клиент/серверных программ:

- отображение (**mapping**) IDL-описаний CORBA-объекта на выбранный язык программирования,
- при этом отображении генерируются специальные программы-заготовки (**stubs**, **skeletons** и **интерфейсы**), которые используются при конструировании программ клиента, сервера и CORBA-объекта,
- компилируются и готовятся исполнительные модули программ клиента и сервера,
- запускается программа регистрации и/или локализации серверов (directory services),
- запускаются или регистрируются программы сервера,
- запускаются на выполнение программы клиента.

Описанная схема имеет очень общий характер и должна выполняться независимо от топологии и количества серверных и клиентских программ и их типа (например, приложений или апплетов).

Первая фаза – **mapping** IDL-интерфейса выполняется специальными программами-компиляторами с различными параметрами, возможностями и функциями в зависимости от выбранного пакета CORBA-реализации:

idltojava – для Java IDL (Sun),
idl2java - для Visibroker 3.3 (Inprise),
idl - для OrbixWeb 3.1 (Iona).

Далее будет дан обзор проведенных исследований различных технологий подготовки и выполнения CORBA-приложений в трех системах, реализованных на языке Java на основе одного примера **HelloWorld** с одинаковым IDL-описанием интерфейса:

Файл **Hello.idl :**

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

Интерфейс содержит описание CORBA-объекта, который должен содержать одну функцию (метод) без параметров с именем **sayHello(),** возвращаемым результатом выполнения которой должна быть текстовая строка.

Задача программы сервера:

- инициализировать CORBA-объект с вышеописанным методом sayHello(),
- образовать брокер заявок (ORB-объект),
- зарегистрироваться под определенным именем в **directory services** (программе локализации) и сообщить ей ссылку **objRef** (object reference) на CORBA-объект,

Задача программы клиента:

- образовать брокер заявок (ORB-объект),
- обратиться к программе локализации **(directory services)** объектов и получить ссылку **objRef** на CORBA-объект,
- используя стандартные средства языка для работы с объектами выполнить оператор обращения к методу объекта

  String hello = objRef.sayHello();

В обзоре будет уделено главным образом внимание тому, каким образом в рассматриваемых разработках реализована вышеуказанная схема, и будут приведены примеры написания программ клиента (приложений и в виде апплетов), сервера и CORBA-объекта.

3. Java IDL (Sun Microsystems Inc.)

Java IDL – так называется реализация архитектуры CORBA средствами языка Java, включенными в виде классов **org.omg** в инструментальный пакет разработчика **JDK 1.2.** В настоящее время **JDK 1.2** разработан компанией **Sun**[4] для **Microsoft Windows 95/98/NT** и **Solaris,** который можно бесплатно получить из **http://www.javasoft.com/.**

Технология применения Java IDL в Microsoft Windows

Предположения:

- пакет **JDK 1.2** инсталлирован в директории **c:\jdk1.2**,
- обеспечен путь доступа к **c:\jdk1.2\bin**,
- для выполнения клиент-приложений в виде апплетов нужно инсталлировать и обеспечить путь доступа к **JRE** (Java Runtime Enviroment),

входящему как дополнительная возможность в JDK 1.2. Это делается в процессе инсталляции основного пакета JDK 1.2.

- все операции по подготовке программ (компиляция и интерпретация) выполняются в **DOS**-режиме (Dos prompt или Dos commander).
- должна быть установлена программа IDL-компилятора **idltojava.exe** (http://www.javasoft.com/). Для простоты можно записать ее в **bin**-директорию JDK.

Примечание 1.

Java IDL предлагает два способа реализации **directory services** для локализации объектов – **Naming service** и **SOR** (Stringified Object Reference),
каждый из которых имеет свои достоинства и недостатки.

Naming service предполагает наличие программы **name**-сервера для обеспечения регистрации и локализации объектов. Эту функцию выполняет программа **tnameserv** из JDK. Метод достаточно универсален, хорошо вписывается в концепцию **directory services**, принятую для распределенных систем. Недостатком метода можно считать отсутствие *автоматического* способа обнаружения **name**-сервера программами клиента и сервера. В рассматриваемом ниже примере демонстрируется именно этот метод.

Механизм **SOR**-методики заключается в формировании текстовой строки **IOR** (Interoperable Object Reference), содержащей ссылку на CORBA- объект (object reference), программой сервера и передаче ее каким-либо способом (через файл или входные параметры) программе клиента. Эта процедура выполняется методами ORB-объекта **object_to_string() и string_to_object** ().

Формирование строки программой сервера:

String ior = orb.object_to_string(helloRef);

Восстановление объекта программой клиента:

org.omg.CORBA.Object obj = orb. (ior);

Метод достаточно надежен в работе, не зависит от реализации **directory services** различными программными пакетами для CORBA и может применяться в этой связи как средство обеспечения интероперабельности. С другой стороны, существуют определенные трудности в проблеме передачи от сервера клиенту строки **IOR**.

Можно привести пример эффективного применения этого метода: программа сервера после формирования IOR-строки генерирует два файла с автоматической передачей этой строки программе сервера:

- командный **bat-**файл для вызова клиент-приложения:

```
@echo off
java Client IOR:000000000000001049444c3a41736572762f413a312e
3000000000010000000000000030000100000000000764637431333000000457000000
0000018afabcafe000000023855a612000000080000000000000000
```

- **HTML-** файл для вызова апплета и передачи ему строки IOR:

```
<html>
<body>
<applet code=AppClient.class codebase=classes width=550 height=550>
<param name=IOR
value="IOR:000000000000001049444c3a41736572762f413a312e300000000001000000
0000000030000100000000000764637431333000000457000000000018afabcafe0000000
23855a612000000080000000000000000">
</applet>
</body>
</html>
```

Замечания: 1. Строка **IOR** при каждом новом вызове программы сервера принимает новое значение.

2. Остается открытой проблема доступа к указанным файлам в реальных распределенных (с разными ЭВМ) системах.

Примечание 2.

Ниже будет рассмотрен пример программирования клиент/серверных программ и методика их подготовки и выполнения. Программы **name-**сервера, клиента и сервера выполняются на одной машине разными процессами. Это ни в коей мере не уменьшает общности исследования CORBA-технологии при выполнении программ клиент/сервера в виде *Java-приложений*, поскольку обслуживание взаимодействия этих процессов обеспечивается протоколом TCP/IP.

Несколько иная ситуация возникает при выполнении клиентской программы в виде Web-приложения (*апплетов*). Тестирование их выполнения на одной машине не означает аналогичности их поведения в случае разнесения моделируемой распределенной системы по разным машинам. Здесь в один клубок сплетаются проблемы одновременной работы Web-сервера, Web-браузеров с их ограничениями на выполнение апплетов, программы name- сервера и собственно серверной программы.

Пример.

В текущей директории **HelloWorld** подготовлены 5 файлов:

Hello.idl – описание интерфейса CORBA-объекта,
HelloClient.java – программа клиент-приложения,
HelloApplet.java - программа апплет-приложения,
Hello.java – серверная программа,
HelloServant.java – программа реализации CORBA-объекта.

Примечание: Иногда программы сервера и CORBA-объекта объединяют в один программный модуль.

Файл **Hello.idl**

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

Файл **HelloClient.java**

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{
  public static void main(String args[])
  {
   try{
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);

      // get the root naming context
      org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references("NameService");

      NamingContext ncRef = NamingContextHelper.narrow(objRef);

      // resolve the Object Reference in Naming
      NameComponent nc = new NameComponent("Hello", "");
      NameComponent path[] = {nc};
      Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

     // call the Hello server object and print results
     String Hello = helloRef.sayHello();
     System.out.println(Hello);
   } catch (Exception e) {
     System.out.println("ERROR : " + e) ;
     e.printStackTrace(System.out);
   }
```

```
    }
  }
```

Файл **HelloApplet.java**

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.awt.*;
import java.util.*;

public class HelloApplet extends java.applet.Applet
{
  String message = "";
  String mess     = "empty string";
  String argHost[];
  public void init() {
   try {
     // create and initialize the ORB
     ORB orb = ORB.init(this, null);
     // get the root naming context
     org.omg.CORBA.Object objRef =
         orb.resolve_initial_references("NameService");
     NamingContext ncRef = NamingContextHelper.narrow(objRef);
     // resolve the Object Reference in Naming
     NameComponent nc = new NameComponent("Hello", "");

     NameComponent path[] = {nc};
     ncRef.resolve(path);

      Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

       // call the Hello server object and print results
       message = helloRef.sayHello();
       mess =message.substring(1, message.length()-1);

   } catch (Exception e) {
     System.out.println("HelloApplet exception: " + e.getMessage());
     e.printStackTrace(System.out);
     }
  }
public void paint(Graphics g)
  {
   g.setColor(Color.black);
   g.drawRect(0,0,106,56);
   g.drawRect(2,2,102, 52);
   g.setColor(Color.green);
   g.fillRect(4,4, 100, 50);
   g.setColor(Color.black);
   g.drawString(mess, 10, 20);
  }
```

```
    }
```

Файл **HelloServer.java**

```
import HelloApp.*;
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloServer {

  public static void main(String args[])
  {
   try{
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);

       System.out.println("Server Hello started!");
      // create servant and register it with the ORB
      HelloServant helloRef = new HelloServant();
      orb.connect(helloRef);

      // get the root naming context
      org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);

      // bind the Object Reference in Naming
      NameComponent nc = new NameComponent("Hello", "");
      NameComponent path[] = {nc};
      ncRef.rebind(path, helloRef);
      //-----------------------------------------------------
      System.out.println("Wait for invocations from clients");

      // wait for invocations from clients
        java.lang.Object sync = new java.lang.Object();
        synchronized (sync) {
            sync.wait();
        }
    } catch (Exception e) {
      System.err.println("ERROR: " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

Файл **HelloServant.java**

```
import HelloApp.*;
import java.io.*;
```

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

class HelloServant extends _HelloImplBase
{
  public String sayHello()
  {
     System.out.println("SayHello");
     return "\nHello world !!\n";
  }
}
```

3.1. Компиляция файла Hello.idl

idltojava -fno-cpp Hello.idl

В результате компиляции в новой директории **HelloApp** (по имени модуля в файле Hello.idl) будут сгенерированы Java-файлы:

Hello.java
HelloHolder.java
HelloHelper.java
_HelloStub.java
_HelloImplBase.java

3.2. Компиляция Java-файлов

Откомпилировать все Java-файлы в текущей директории **HelloWorld** и сгенерированные файлы в **HelloApp** можно одной командой вызова компилятора:

javac *.java

Примечание: компилировать можно и отдельные файлы, например:
javac HelloServant.java.

3.3. Запуск name-сервера

Функцию name-сервера выполняет программа **tnameserv.exe** из JDK.
Запуск:
tnameserv [-ORBInitialPort 1050]
Запуск программы без параметров означает задание номера порта по умолчанию – 900.

Программа отслеживает по заданному порту запросы серверных программ на регистрацию и запросы клиентских программ на обнаружение и подключение к серверным программам.

Программы-серверы идентифицируются своим именем, которое задается при регистрации оператором:

NameComponent nc = new NameComponent("Hello", "");

Запрос клиентской программы выполняется таким же оператором:

NameComponent nc = new NameComponent("Hello", "");

Из примера видно, что сервер зарегистрирован под именем **Hello**.

Номер порта, с которым работает name-сервер в программах клиента и сервера задается при инициализации ORB-объекта (брокера) одинаковым оператором:

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

args – массив строк:

args[0] = "-ORBInitialPort"
args[1] = "номер порта"

Если массив пуст, устанавливается значение порта 900.

3.4. Запуск серверной программы

java HelloServer или
java HelloServer –ORBInitialHost js.jinr.ru –ORBInitalPort 1050

Параметр **ORBInitialHost** задает имя машины, на которой запущена программа tnameserv – name-сервер, а **ORBInitalPort** – номер порта.

В первом случае в качестве машины, обслуживающей сервер имен, выбирается **localhost** с портом 900, во втором случае: сервер имен обслуживается на машине **js.jinr.ru** и номер порта задан как 1050.

Программа сервера выполняет следующие операции:

- инициализацию брокера:
  ORB orb = ORB.init(args, null);
- инициализацию CORBA-объекта и подключение его к брокеру:
  HelloServant helloRef = new HelloServant();
  orb.connect(helloRef);
- поиск name-сервера:
  org.omg.CORBA.Object objRef =
  orb.resolve_initial_references("NameService");
  NamingContext ncRef = NamingContextHelper.narrow(objRef);

- регистрацию сервера:
  NameComponent nc = new NameComponent("Hello", "");

- передачу **name**-серверу ссылки (object reference) на CORBA-объект:
  NameComponent path[] = {nc};
  ncRef.rebind(path, helloRef);

- переход в режим ожидания:
  java.lang.Object sync = new java.lang.Object();
  synchronized (sync) {
  sync.wait();
  }

3.5. Запуск программы клиента

java HelloClient или
java HelloClient –ORBInitialHost js.jinr.ru –ORBInitialPort 1050

Значения параметров такие же, как и при запуске серверной программы.
Программа клиента выполняет следующие операции:

- инициализация брокера и обращение к name-серверу выполняются такими же операторами, как и в программе сервера,
- запрос ссылки (**object reference**) на CORBA-объект:
  NameComponent path[] = {nc};
  Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));
- обращение к методу CORBA-объекта:
  String Hello = helloRef.sayHello();
- печать полученного результата:
  System.out.println(Hello);

3.6. Выполнение апплетов в appletviewer

Для выполнения апплета любым браузером необходимо подготовить HTML-файл с вызовом апплет-класса.

Пример: файл **hello.html.**

```
<HTML>
<HEAD>
  <TITLE>Java IDL: Running HelloApplet</TITLE>
</HEAD>
<BODY>
<APPLET CODE=HelloApplet.class  WIDTH=200  HEIGHT=200>
</APPLET>
</BODY>
</HTML>
```

Единственным отличием программ клиента в виде апплета или приложения является способ формирования ORB-объекта. В апплете это выполняется оператором

ORB orb = ORB.init(this, null);

Запуск апплет-клиента:

appletviewer hello.html

3.7. Выполнение CORBA-апплетов Web-браузерами

Серьезной проблемой до недавнего времени оставалось выполнение CORBA-апплетов широко распространенными Web-браузерами Internet Explorer (**IE**) и/или Netscape Communicator (**NC**). Главная причина – отсутствие (полное в IE или частичное в NC) поддержки протоколов CORBA, реализованных в виде классов. В настоящее время, используя механизм **plugins** браузеров, удается подключить классы **omg.org**, установленные в JRE из JDK (см. выше) для выполнения их в апплетах. Для этого классы апплетов вызываются не стандартным для HTML способом (через теги **<applet>**), а специальными средствами, разными для **IE** и **NC**.

Вызов апплетов с включением классов JDK 1.2 в Netscape 4.0:

<EMBED type="application/x-java-applet;version=1.2"
java_CODE = HelloApplet.class
WIDTH = 200 HEIGHT =
pluginspage="http://java.sun.com/products/plugin/1.2/plugin-install.html">
<NOEMBED>
alt="Your browser understands the <APPLET> tag but isn't running the applet, for some reason."
Your browser is completely ignoring the <APPLET> tag!
</NOEMBED>
</EMBED>

Вызов апплетов с включением классов JDK 1.2 в Internet Explorer 4.0:

<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 300 HEIGHT = 300
codebase="http://java.sun.com/products/plugin/1.2/jinstall-12-win32.cab#Version=1,2,0,0">
<PARAM NAME = CODE VALUE = HelloApplet.class >
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
<PARAM NAME = model VALUE =models/HyaluronicAcid.xyz>
</OBJECT>

Естественно, эти пугающие коды вызывает определенные неудобства для разрабочиков Web-страничек с включением CORBA-апплетов. Но, к сожалению, это - следствие противостояния конкурирующих компаний. Можно придумать более сложный пример HTML-файла с двумя одновременными вызовами апплетов: теги **<object>** не воспринимаются Netscape браузерами, а теги **<embed>** для вызова апплетов Netscape браузерами для сокрытия от Internet Explorer заключаются в теги комментария **<comment>**.

Пример:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 300 HEIGHT = 300
 codebase="http://java.sun.com/products/plugin/1.2/jinstall-12-win32.cab#Version=1,2,0,0">
<PARAM NAME = CODE VALUE = HelloApplet.class >
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
<PARAM NAME = model VALUE =models/HyaluronicAcid.xyz>
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2" java_CODE = HelloApplet.class
 WIDTH = 200 HEIGHT = 320 pluginspage="http://java.sun.com/products/plugin/1.2/plugin-
install.html">
<NOEMBED>
</COMMENT>
alt="Your browser understands the &lt;APPLET&gt; tag but isn't running the applet, for some
reason."
        Your browser is completely ignoring the &lt;APPLET&gt; tag!
</NOEMBED></EMBED>
</OBJECT>
```

Возможны и другие реализации разделения вызовов апплетов. Например, можно использовать **JavaScript** для программного распознавания браузера, загрузившего текущую HTML-страницу, и выполнить разные алгоритмы вызова апплета.

Существует еще один способ применения апплетов стандартным браузером NC 4.0, без привлечения механизма **plugins,** который требует обязательной инсталляции на машине клиента JDK 1.2. Автор применял эту методику еще до появления этого средства. Для этого используются встроенные в NC 4.0 классы CORBA – Visigenic ORB и механизм SOR для передачи строки IOR.

Пример фрагмента программы сервера и клиента:

Сервер:

```
ORB orb = ORB.init(args, null);
 HelloServant helloRef = new HelloServant();
 orb.connect(helloRef);
 String ior = orb.object_to_string(helloRef);

 String batFile = "Client.bat";
 String htmlFile = "Client.html";

 FileOutputStream fosb = new FileOutputStream(batFile);
 FileOutputStream fosh = new FileOutputStream(htmlFile);
 PrintStream ps = new PrintStream(fosb);
 PrintStream psh = new PrintStream(fosh);

  // Генерация в текущей директории HTML-файла для вызова апплета
  ps.print("<html><body>\n");
  ps.print("<applet code=AppClient.class codebase=classes width=550
                                                          height=550>\n");
  ps.print("<param name=IOR ");
  ps.print("value=\"" + ior + "\">\n");
  ps.print("</applet>\n");
  ps.print("</body></html>");
  ps.close();

  // Генерация bat-файла для вызова клиент-приложения
  psh.print("@echo off  \n java HelloClient " + ior);
  psh.close();
```

Клиент-приложение:

```
ORB orb = ORB.init(arg, null);
String ior = args[0];
org.omg.CORBA.Object obj = orb.string_to_object(ior);
Hello helloRef = HelloHelper.narrow(obj);
```

Клиент-апплет:

```
String ior = getParameter("IOR");
ORB orb = ORB.init(this, null);
```

org.omg.CORBA.Object obj = orb.string_to_object(ior);
Hello **helloRef** = HelloHelper.narrow(obj);

3.8. Выводы

В Java IDL представлен наиболее простой способ для разработок распределенных систем с CORBA-архитектурой:

- представлен собственный метод отображения (mapping) описаний IDL-языка на Java – компилятор **idltojava**, достаточно простой в применении;
- имеются все необходимые средства для создания **брокеров** заявок – классы org.omg.CORBA.* в составе JDK 1.2;
- средства локализации объектов (**directory services**) представлены двумя видами – Naming Service и Stringified Object Reference;
- достаточно просто реализуется режим **callback**;
- имеется возможность применения клиент-приложений в виде **апплетов** в среде WWW.

Немаловажным обстоятельством, сделавшим популярным эту разработку, является свободное распространение пакета JDK 1.2 для платформ с операционными системами Microsoft Windows 95/98/NT и Solaris.

К слабостям предложенной системы можно отнести недостаточный сервис. Это, в первую очередь - отсутствие *автоматической локализации* объектов: пользователю клент/серверных программ надо либо указывать точный сетевой адрес name-сервера, либо проявлять незаурядную изобретательность при выборе метода SOR для этой цели. К тому же полностью отсутствуют программные средства (утилиты) для наблюдения и контроля в распределенной системе. В отличие от других аналогичных систем документация представлена в незначительном объеме, что, впрочем, можно объяснить, с одной стороны, простотой и прозрачностью ее применения, а с другой – отсутствием дополнительных нагромождений, не всегда способствующих повышению ее эффективности.

Java IDL можно было бы рекомендовать для начинающх изучение и применение Java-технологии в распределенных системах.

4. VisiBroker Java 3.3 (Inprise Corp.)

Одним из серьезных разработчиков программного обеспечения для распределенных систем, включая CORBA-архитектуру, является компания **Inprise Corp**[5]. В конце 1998 года компания представила новый коммерческий продукт для CORBA – пакет **VisiBroker 3.3** для объектно-ориентированных языков программирования C++ и Java. Пакет для Java-реализаций называется VisiBroker Java 3.3 или же **VBJ33**. Компания предоставляет для ознакомления и тестирования **evaluate**-версию данного пакета. Пакет содержит большое число программ из обязательного набора для CORBA-технологии (категория **Directory Services**) и полезных утилит, в том числе и для наблюдения функционирования распределенной системы (категория **Monitor transaction**).

Пакет разработан для платформ с операционными системами:
Solaris 2.5, 2.5.1, 2.6
Windows 95/98, NT 4.0, NT 3.5.1

HP-UX 10.20, 11.0
IRIX 6.2, 6.3, 6.4
AIX 4.1, 4.2
Digital Unix 4.0

Для выполнения программ VisiBroker Java 3.3 требуется установка на машине Java-комплекта (желательно JDK 1.1.8).

В данном разделе будут рассмотрены два варианта применения технологии VisiBroker 3.3 для **Microsoft Windows** на простом примере Hello.idl из предыдущего раздела:

- запуск программы сервера вручную,
- автоматический запуск сервера по запросу клиента.

Технология применения VBJ33 в Microsoft Windows

При инсталляции пакета **VisiBroker Java 3.3** устанавливаются переменные окружения **VBROKER_ADM, OSAGENT_PORT**. Переменные используются при выполнении программ VBJ33, значения их могут переопределяться DOS- командой SET или параметрами при вызове программ.

Предположения:

- пакет **JDK 1.1.8** инсталлирован в директории **c:\jdk1.1.8**,
- пакет VisiBroker Java 3.3 инсталлирован в директории **c:\vbj33,**
- обеспечены пути доступа к **c:\jdk1.1.8\bin** и **c:\vbj33\bin,**
- все операции по подготовке программ (компиляция и интерпретация) выполняются в **DOS**-режиме (Dos prompt или Dos commander).

В данном случае будут установлены значения переменных окружения:
VBROKER_ADM=C:\vbj33\adm,
OSAGENT_PORT=14000,

Примечания: 1. Часть программ из пакета VBJ33 используют Java-программы из JDK, в том числе и IDL- компилятор. При неправильной инсталляции или примененииии JDK есть риск получить не очень понятную диагностику.

2. Обработка Java-программ должна выполняться препроцессорами:

- **vbjc** – для компиляции Java программ,
- **vbj** – для их выполнения (интерпретации).

Надо отметить ряд необходимых и полезных программ из VBJ33:

java2idl – IDL компилятор,
vbjc и **vbj** – препроцессоры для Java программ,
osagent – Smart Agent, применяется для локализации объектов,
oadj – Object Activation Daemon, применяется для автоматического запуска программы сервера по запросу клиента,
osfind – утилита для выдачи отчета о состоянии сервисных служб (Smart Agent, oadj и т. д.) и объектов в распределенной системе,
oadutil – утилита для администрирования (регистрации) объектов.

Пример.

В текущей директории **HelloWorld** подготовлены 5 файлов:

Hello.idl – описание интерфейса CORBA-объекта,
HelloClient.java – программа клиент-приложения,
HelloApplet.java – программа клиент-апплет,
HelloServer.java – серверная программа,
HelloServant.java – программа реализации CORBA-объекта.

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("SayHello");
     return "\nHello world !!";
    }
  }
```

Файл **HelloApplet.java**

```
import java.awt.*;
import java.util.*;
import java.applet.Applet;

public class HelloApplet extends java.applet.Applet {
  String message = "";
  String mess = "empty string";
  org.omg.CORBA.ORB orb;
  public void init() {
   try {
     // create and initialize the ORB
        orb = org.omg.CORBA.ORB.init(this,null);
        HelloApp.Hello helloRef =
              HelloApp.HelloHelper.bind(orb, "HelloServer");
        message = helloRef.sayHello();
        mess =message.substring(1, message.length()-1);
   } catch (Exception e) {
     System.out.println("HelloApplet exception: " + e.getMessage());
     e.printStackTrace(System.out);
    }
  }
public void paint(Graphics g){
    g.setColor(Color.black);
    g.drawRect(0,0,106,56);
    g.drawRect(2,2,102, 52);
    g.setColor(Color.green);
    g.fillRect(4,4, 100, 50);
    g.setColor(Color.black);
   g.drawString(mess, 10, 20);
  }
}
```

4.1. Компиляция файла Hello.idl

idl2java Hello.idl

В результате компиляции в новой директории **HelloApp** (по имени модуля в файле Hello.idl) будут сгенерированы Java-файлы:

Hello.java
HelloHolder.java
HelloHelper.java
HelloOperations.java
_st_Hello.java
_HelloImplBase.java

_tie_Hello.java
_example_Hello.java

4.2. Компиляция Java-файлов

Откомпилировать все Java-файлы в текущей директории **HelloWorld** и сгенерированные файлы в **HelloApp** можно одной командой вызова компилятора:

vbjc *.java

Примечание: компилировать можно и отдельные файлы, например:
vbjc HelloServant.java.

4.3. Локализация объектов

Локализацию объектов и координацию клиент-серверных приложений выполняет программа Smart Agent (**osagent.exe**). В локальной Internet-сети должен быть запущен как минимум один процесс с этой программой. Поиск программы Smart Agent серверами и клиентами выполняется автоматически с использоаванием широковещательного UDP-запроса (brodcasting messages).

Запуск Smart Agent:
osagent [-C] [-p UDP_port] [-verbose]
Значения параметров:

- **C** задается вWindows NT,
- **UDP_port** переопределяет значение порта прослушивания (по умолчанию равное 14000),
- **verbose** устанавливает режим записи инфомационных и диагностических сообщений в файл <VBROKER_ADM>\log\osagent.log

4.4. Запуск сервера.

Как выше уже отмечалось, будут продемонстрированы два подхода к запуску программы сервера (ручной и автоматический).

4.5. Ручной запуск программы сервера

vbj [-D OaipAddr <hostname | ip-address>] [-D OAport <port number>]
<Server program>

Параметры **OAipAddr** и **OAport** устанавливают в явном виде сетевой адрес машины и номер порта прослушивания для программы **osagent.**

Для нашего примера эта процедура выглядит как:

vbj HelloServer

4.6. Автоматический запуск программы сервера

Для выполнения работ по автоматическому запуску серверной программы (в том числе и для регистрации) всегда должен находиться в работе **Object Activation Daemon**. Поскольку эта программа будет использоваться для процедур регистрации и вызова серверных программ, необходимо перед ее запуском установить в качестве значений переменной CLASSPATH пути доступа к серверным программам (классам). Удобно объединить эти операции в командном файле **oadj.bat:**

```
@echo 0ff
set CLASSPATH=%CLASSPATH%;C:\vbj33\HelloWorld
oadj.exe
```

Для регистрации сервера применяется утилита **oadutil** с параметрами:

oadutil reg –r <repository ID> -o <object name> -java <class name>

Значения параметров:

Простейший способ определения значения параметра **repository ID** – вызвать утилиту **osfind** (см. ниже) при запущенной ручным способом программе сервера,
object name - имя, которое присвоено серверу самой программой,
class name - имя откомпилированной программы сервера (класса).

Для нашего примера вызов утилиты выглядит следующим образом:

oadutil reg –r IDL:HelloApp/Hello:1.0 –o HelloServer –java HelloServer

Если регистрация прошла нормально, на консоль будет выдано сообщение типа

```
Completed registration of repository_id    =  IDL:HelloApp/Hello:1.0
object_name      =  HelloServer
reference data   =
path_name        =  vbj
activation_policy =  SHARED_SERVER
args             =  (length=1)[HelloServer; ]
env              =  NONE
for OAD on host 159.93.17.196
```

Регистрация сервера выполняется один раз, независимо от числа обращений к нему. Удаление из регистрации выполняется той же утилитой с параметром **unreg.**

Например:

oadutil unreg –r IDL:HelloApp/Hello:1.0 –o HelloServer

Просмотреть состояние зарегистрированных объектов можно командой:

oadutil list

Замечание: все виды работ утилиты **oadutil** выполняются при запущенных программах **oadj** и **osagent.**

4.7. Запуск клиент-приложения

Запуск клиент-программы выполняется так же и с теми же параметрами, как и для программы сервера:

vbj [-D OaipAddr <hostname | ip-address>] [-D OAport <port number>]
<Client program>

Для нашего примера это будет:

vbj HelloClient.

Итак, полная последовательность запуска клиент-серверной пары

при ручном запуске сервера:

- **osagent** – запуск в локальной сети **Smart Agent**,
- **vbj HelloServer** – запуск серверной программы,
- **vbj HelloClient** – запуск клиентской программы;

при автоматическом запуске сервера (после проведенной регистрации):

- **osagent** – запуск в локальной сети **Smart Agent**,
- **oad.bat** – запуск программы **Object Activation Daemon,**
- **oadutil reg –r IDL:HelloApp/Hello:1.0 –o HelloServer –java HelloServer** – регистрация сервера,
- **vbj HelloClient.**

4.8. Выполнение клиент-приложений в виде апплетов

Как и в предыдущем разделе, рассмотрим проблемы применения апплетов двумя способами – просмотрщиком **appletviewer** из JDK и стандартными Web- браузерами **Netscape Communicator 4.0** и/или **Internet Explorer 4.0.**

Применение **appletviewer** обычно используется для проверки принципиальной работоспособности апплетов.

Координацию выполнения апплетов по технологии VisiBroker 3.3 выполняет программа **gatekeeper** – IOOP Proxy Server. Отметим также, что в случае применения Web-браузера программа **gatekeeper** выполняет также и туннелирование HTTP-протокола, т. е. функцию Web-сервера.

Для надежной работы программы **gatekeeper** надо обеспечить ей нахождение необходимых файлов и классов. Для этого удобно составить командный файл **gatekeeper.bat** типа

```
@echo off
 set PATH=%PATH%;c:\vbj33\HelloWorld
set CLASSPATH=%CLASSPATH%;c:\vbj33\lib\vbjorb.jar;c:\vbj33\HelloWorld
gatekeeper.exe
```

Для выполнения апплета любым способом необходимо подготовить HTML-файл с вызовом Java-класса.

Пример файла **hello.html:**

```
<HTML>
<TITLE> Running HelloApplet</TITLE>
</HEAD>
<BODY>
<APPLET CODE=HelloApplet.class  WIDTH=200  HEIGHT=200>
<param name=org.omg.CORBA.ORBClass value=com.visigenic.vbroker.orb.ORB>
</APPLET>
</BODY>
</HTML>
```

В любом случае (для appletviewer или браузеров) , для запуска апплет-клиента должны быть в работе сервисные службы -

при постоянно запущенной программе сервера:

- **osagent**
- **gatekeeper.bat**
- **vbj HelloServer**

для автоматического вызова сервера:

- **osagent**
- **gatekeeper.bat**
- **oadj.bat**

Выполнение апплетов просмотрщиком appletviewer

Перед вызовом **appletviewer** необходимо установить в переменной окружения **CLASSPATH** доступ к **ORB-**классам из пакета **VisiBroker 3.3**. Все это удобно объединить в командном файле **applet.bat**:

```
@echo off
set CLASSPATH=.;c:\vbj33\lib\vbjorb.jar
appletviewer $1
```

Запуск клиент-апплета в таком случае выглядит как:

applet.bat hello.html

Выполнение апплетов браузерами NC 4.0 и IE 4.0

При запуске браузера должно быть задан URL: http://host:15000/<html файл>.

Для рассматриваемого примера это будет: **http://js.jinr.ru:15000/hello.html**

4.9. Утилиты для распределенных систем

В пакете VisiBroker Java 3.3 имеется, кроме выше упомянутых, ряд полезных программ для администрирования распределенной системы – **locserv, irep, osfind.** Как ранее упоминалось, для формирования параметров утилиты **oadutil,** можно воспользоваться программой **osfind** для выдачи отчета о доступном сервисе и объектов в распределенной системе.

Пример выдачи утилиты **osfind:**

```
osfind: Found one agent at port 14000
        HOST: wnct196.jinr.dubna.su

osfind: Found 1 OADs in your domain
        HOST: wnct196.jinr.dubna.su

osfind: Following are the list of Implementations registered with OADs.
        HOST: wnct196.jinr.dubna.su
        REPOSITORY ID: IDL:HelloApp/Hello:1.0
        OBJECT NAME: HelloServer

osfind: Following are the list of Implementations started manually.
        HOST: wnct196.jinr.dubna.su
```

REPOSITORY ID: IDL:visigenic.com/gatekeeper/AliasManager:1.0
OBJECT NAME: IIOP Gate-Keeper
REPOSITORY ID: IDL:HelloApp/Hello:1.0
OBJECT NAME: Server

4.10. Выводы

Предложенный компанией Inprise пакет VisiBroker 3.3[1] является одним из наиболее проработанных для распределенных систем с CORBA-архитектурой. Как видно из предыдущего, пакет предназначен для функционирования на ряде платформ и содержит кроме обязательного набора программного обеспечения также ряд утилит для администрирования распределенной системы. Пакет содержит необходимую документацию в формате HTML, кроме того, можно из Web-сайта компании получить более специализированную документацию для углубленного изучения.

К недостатку системы можно отнести все же некоторые досадные умалчивания, непоследовательность и несистематичность в документации, приводящую даже к противоречиям. Так, противоречивы рекомендации в двух источниках (Programmer's Guide и Installation and Administration Guide) для выполнения клиент/серверных приложений, находящихся в различных локальных сетях (доменах).

5. OrbixWeb 3.1 (Iona Technologies Inc.)

Пакет OrbixWeb 3.1 является коммерческим продуктом. Для изучения и тестирования можно бесплатно получить [6] **evaluate**-версию программ и документацию на 30 дней.

Предположения:

- В директории **c:\jdkl.l.8** инсталлирован **JDK 1.1.8**.
- Пакет **OrbixWeb 3.1** инсталлирован в директории **C:\Iona\Orbix\Veb3.1c.**
- Пути доступа к **C:\Iona\Orbix\Veb3.1c\bin** и **c:\jdkl.l.8\bin** прописаны в **autoexec.bat.**
- Все запуски утилит и клиент/серверных программ выполняются в DOS-режимах (DOS prompt или DOS commander).
- В текущей директории подготовлены 4 файла:

Hello.idi - описание интерфейса **CORBA**-объекта
Clientjava - клиентская программа
Server.java - серверная программа
HelloServant.java - программа реализации **CORBA**-объекта

Файл **HelloClient.java:**

```
package helloDemo;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.SystemException;
import org.omg.CORBA.ORB;
public class HelloClient {

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

[1] В настоящее время компания выпустила новый продукт семейства VisiBroker 4.x для Java и C++.

```
String hostName = null;
Hello helloRef = null;

if (args.length < 1) {
      System.out.println ("Usage : Client [<hostname>]\n");
 System.exit (1);
}
else {
/*
 * Get the host machine name where the activator is running.
 */
 hostName = new String (args[0]);
}
 /*
 * Initalise the ORB for an OrbixWeb application.
 */
ORB.init (args,null);
 /*
 * Establish a CORBA connection with the grid server and
 * return a proxy for it.
 * The bind call will cause the activator (orbixd/orbixdj) to
 * launch the grid server, and enable it to accept remote requests.
 */
try {
 helloRef = HelloHelper.bind (":HelloServer", hostName);
}
/*
 * If the bind should fail the exception error will
 * be caught in the catch block and displayed.
 */
catch (SystemException ex) {
 System.out.println ("Exception caught during bind : " + ex.toString());
 System.exit (1);
}
String hello = helloRef.sayHello();
 System.out.println(hello);
}

}
```

Файл **HelloServer.java:**

```
package helloDemo;
import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.SystemException;
//--------------------------------------
public class HelloServer {
 public static void main(String[] args) {
 Hello helloRef=null;
// Initalise OrbixWeb for an application
```

```
   ORB orb = ORB.init (args,null);
try {
    // Create an implementation of the grid server object
    helloRef = new HelloServant("HelloServer");

   /*
    * The impl_is_ready call takes the name of the server as registered
    * with the activator (orbixd,orbixdj).
    * It indicated to OrbixWeb that the server has been initalised and
    * is ready to receive operation requests on it objects, in this case the
    * gridImpl object.
    */
     _CORBA.Orbix.impl_is_ready ("HelloServer");
    System.out.println ("Shutting down helloServer...");
    /*
     * Indicate to OrbixWeb that all invocations are finished.
     */
     //orb.shutdown (helloRef);
   }
   catch (SystemException se) {
     System.out.println ("Exception during creation of HelloServant" + se.toString());
     System.exit (1);
   }
  System.out.println ("HelloServer exiting....");
  }
}
```

Файл **HelloServant.java:**

```
package helloDemo;
import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.SystemException;

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

5.1. Компиляция файла Hello.idl

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

Для **OrbixWeb 3.1** удобнее файл представить в виде:

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

Компиляция файла:
idl -jP helloDemo Hello.idi
helIoDemo - название пакета, как указано в строке package helloDemo в клиент/серверных программах.

При этом в новой директории **java_output/helloDemo** образуются файлы:
HelloPackage - директория
Hello.java
HelloHolder.java
HelloHelper.java
_HelloStub.java
_HelloOperations.jjava
_HelloSkeleton.java
_tie_Hello.java
_HelloImplBase.java

5.2. Компиляция java-файлов

Есть несколько вариантов вызова компилятора javac для java-программ:
- препроцессором **owjavac**, который вызывает javac-компилятор с инсталляционными параметрами;
- "ручной" вызов javac-компилятора, через bat-файлы.

Вызов препроцессора **owjavac** равносилен вызову компилятора javac с параметрами:

```
javac-classpath "C:\Iona\Orbix\Veb3.1c\classes;C:\jdkl.l.8\lib\classes.zip"
–d "C:\Iona\OrbixWeb3.1c\classes"
```

При формировании bat-файлов можно сократить ручной способ задания параметров и вызывать препроцессор **owjavac** или же непосредственно компилятор **javac**. К примеру, такой файл comp.bat мог бы иметь вид:

```
@echo off
set PKG_NAME=helloDemo
owjavac *.java java_output\%PKG_NAME%\*.java
```

или

```
@echo off
set PKG_NAME=helloDemo
javac -classpath "C:\Iona\OrbixWeb3.1c\classes;C:\jdkl.l.7\Iib\classes.zip"
-d "helloDir" *.java java_output\%PKG NAME %\*.java
```

Компиляция всех (исходных и сгенерированных) **java-**программ:
- вызовом препроцессора:
  owjavac *.java java_output\helloDemo*.java

- bat-файлом в текущей директории:

comp.bat

В первом случае результаты трансляции будут записаны в директорию, указанную параметром **-d,** т.е. в **C:\Iona\OrbixWeb3.1c\classes,** что не всегда удобно. Если не указывать этот параметр (в случае непосредственного выова компилятора javac), классы будут записаны в текущую директорию. Вызов препроцессора с перекрытием этого параметра:

owjavac -d helloDir *.java java_output\helloDemo*.java

При этом в указанной директории **helloDir** будет образована поддиректория **helloDemo** (согласно объявлению package helloDemo в java-программах), В которую будут записаны все классы, как результат трансляции. Выше рассмотрены варианты одновременной трансляции всех программ, но на практике в прцессе отладки программ чаще возникает необходимость компиляции лишь модифицированных модулей. Для этой цели полезным было бы иметь набор различных командных **(bat)** файлов, например **cjavac.bat** для компиляции одиночного модуля:

```
@echo off
set PKG_NAME=helloDemo
javac -classpath "C:\Iona\OrbixWeb3.1c\classes;C:\jdkl.l.8\lib\classes.zip"
-d "helloDir" %1
```

Вызов компилятора:

cjavac Client.java

В любом случае мы должны получить следующую структуру:

```
[helloDir]
        [helloDemo]
                Client.class
                HelloServant.class
                Server.class
                HelloImplBase.class
                _HelloOperations.class
                _HelloSkeleton.class
                _HelloStub.class
                _tie_Hello.class
                Hello.class
                HelloHelper.class
                HelloHolder.class
[java_output]
        [helloDemo]
                [HelloPackage]
                Hello.java
                HelloHolder.java
                HelloHelper.java
                _HelloStub.java
                _HelloOperations.java
                _HelloSkeleton.java
                _tie_Hello.java
```

_HelloImplBase.java

hello.idl
Client.java
Server. java
HelloServant.java

5.3. Запуск OrbixWeb java-демона

Программа типа **daemon** (демон) **orbixdj** запускается на все время работы для автоматической активизации серверных программ, зарегистрированных в базе данных, при вызове их клиентскими программами

start orbixdj.bat

или командой **OrbixWeb Java Daemon** из меню инсталлированного пакета.

5.4. Регистрация сервера

Следующим этапом подготовки программ к запуску является регистрация серверной программы в базе данных **OrbixWeb.** Это выполняется утилитой **putit.**

Замечание: необходимо убедиться при запуске "демона" **orbixdj,** что обеспечен будет его доступ к программе сервера. Для этого необходимо обеспечить пути доступа (переменная PATH) к директории helloDemo, где находится программа сервера Server.class либо запускать программу демона из соответствующей директории:

```
set PATH=%PATH%;c:\....\demoDir
start orbixdj
```

или

```
cd demoDir
start orbixdj
```

Для регистрации сервера надо выполнить последовательность команд:

```
call putit -j HelloServer helloDemo.Server
call chmodit HelloServer i+all
call chmodit HelloServer 1+all
```

HelloServer - имя сервера, заданное программой **Server.java** в строке:
helloRef = new HelloServant("HelloServer") ;
helloDemo - название директории (из строки **package helloDemo;),** в которой находится программа **Server.class,**
Server - имя программы **Server.class**

Имеет смысл подготовить соответствующий командный **bat-**файл **put.bat:**

```
@echo off
set PKG NAME=helloDemo
set SRV_NAME=HelloServer
call putit -j %SRV_NAME% %PKG_NAME%. Server
call chmodit %SRV_NAME% i+all
call chmodit %SRV NAME% 1+all
```

5.5. Запуск клиента

Клиентской программой является файл **Client.class** из директории

helloDir/helloDemo.

Запуск выполняется препроцессором **owjava,** который вызывает java-интерпретатор с параметрами:

```
c:\JDK1.1.8\bin\java.exe -classpath
"c:\Iona\OrbixWeb3.lc\classes;c:\JDK1.1.8\lib\classes.zip'
-DOrbixWeb.config="c:\Iona\OrbixWeb3.lc\classes\OrbixWeb.properties"
```

Вызов клиента:

```
cd helloDir
owjava  -echo helloDemo.Client
```

6. Заключение.

Описанные в работе исследования проводились автором несколько лет назад и были вновь востребованы в связи с новыми задачами. В соответствии с Соглашением о сотрудничестве ОИЯИ и Исследовательского центра Россендорф (FZR, Германия) автором был выполнен ряд исследований, главное направление которых – выбор, анализ и опробование новых технологий для использования их в создаваемых информационных распределенных системах, в частности – для развития информационной инфраструктуры FZR. Достаточно мощным стимулом для таких исследований явилось появление и практическое использование нового поколения средств хранения данных – СУБД Oracle 8i v.1.1.7 со встроенной поддержкой Java-технологий, включая средства для сетевого доступа к данным на основе CORBA-технологий: ORB Visibroker и компонентной, совместимой с CORBA, объектной модели EJB (Enterprise JavaBeans).

Литература

1. Object Management Group. http://www.omg.org/.
2. А. Цимбал. Сравнительный анализ технологий CORBA и COM . http://www.corbadev.kiev.ua/library/art2.html.
3. Н. Дубова. COM или CORBA? Вот в чем вопрос. http://www.gamepro.ru/os/1999/03/06.htm.
4. Sun Microsystems, Inc. http://www.sun.com/.
5. Borland Software Corporation. http://www.inprise.com/.
6. IONA Technologies . http://www.iona.com/.
7. Java Station. http://dbserv.jinr.ru/js/.

Получено 15 марта 2002 г.

Галактионов В. В.
Р10-2002-63

Java-технологии в распределенных системах
с CORBA-архитектурой

В соответствии с Соглашением о сотрудничестве ОИЯИ и исследовательского центра FZR (Россендорф, Германия) автором выполнен ряд исследований, главное направление которых — выбор, анализ и опробование новых технологий для использования их в создании информационных распределенных систем, в частности для организации доступа к реляционным базам данных. В работе приведены результаты этих исследований.

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2002

Перевод автора

Galaktionov V. V.
Р10-2002-63

Java Technologies for Distributed Systems Based
on Use of CORBA Architecture Oriented Middleware

According to JINR Agreement with FZR (Rossendorf, Germany), author performed some investigations, concerning use of Java/CORBA technologies for creation of distributed information systems (in particular, based on access to relational data bases). Result of these researches (choice, analysis and practical testing of these technologies possibilities) is the main topic of this publication.

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 16.04.2002.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 2,7. Уч.-изд. л. 3,1. Тираж 300 экз. Заказ № 53238.

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