**Лекция 8**

Приложения, рассматриваемые в предыдущих разделах, строились исходя из предположения о том, что интерфейсы объектов заранее известны. Собственно, иначе и быть не могло, поскольку именно эти интерфейсы и описывались на IDL перед началом разработки приложений.

Однако CORBA позволяет строить приложения, оперирующие объектами, интерфейсы которых неизвестны на момент компиляции приложения. Более того, возможно создание объектов, интерфейс которых будет изменяться в процессе их жизни (под интерфейсом мы понимаем совокупность методов объекта, которые может вызывать клиент). Данные возможности активно применяются при решении задач интеграции приложений, когда CORBA выступает в качестве "клея", связующего компоненты, которые выполнены в различных технологиях.

Соответствующие элементы технологии CORBA носят названия *Dynamic Invocation Interface (DII)* и *Dynamic Skeleton Interface (DSI)*.

С помощью первой технологии ( DII ) клиенты могут формировать динамические вызовы методов объектов - формировать не в момент компиляции, а в процессе работы.

С помощью второй технологии ( DSI ) серверные объекты могут регистрировать в системе новые методы, которые они будут реализовать, или изменять спецификацию старых. Для написания объектов с использованием этих технологий IDL уже не применяется, и многие из задач, которые раньше решались автоматически сгенерированным кодом, при таком подходе придется решать самостоятельно.

В качестве примера, на котором будет проиллюстрировано использование этих технологий, возьмем уже использованный ранее "Первый пример" из данного раздела. Перепишем клиентский класс таким образом, чтобы он формировал динамические вызовы к серверу (т.е. не использовал клиентскую заглушку), а серверный - так, чтобы он сам регистрировал реализуемые им методы в системе (т.е. не использовал автоматически сгенерированный скелетон).

Сами сигнатуры методов серверного объекта оставим без изменений. Соответственно, для реализации этого примера IDL -файл с описанием интерфейсов не нужен.

**Класс BillingServiceImpl**

Прежде всего, следует отметить, что поскольку теперь невозможно использовать автоматически сгенерированные классы (их попросту нет), серверный класс должен наследовать от **DynamicImplementation** и перекрывать его метод **invoke**,который вызывается при любом обращении клиента.

Диспетчеризацию сообщений теперь придется выполнять вручную, соответствующий код реализован в методе **invoke**.

Итак, серверный класс будет содержать огромный метод **invoke**,в котором будет определяться то, какой метод сервера хочет вызвать клиент, и будут извлекаться параметры, переданные клиентом. Соответствующие методы, реализующие операции с картами и изменяющие баланс, полностью аналогичны рассмотренным ранее в предыдущем примере.

1 package com.asw.corba.ex3;

2

3 import java.util.\*;

4 import org.omg.CORBA.\*;

5

6 class BillingServiceImpl extends DynamicImplementation {

7

8 static String[] myIDs = { "IDL:JavaIDL/DSIBillingService:1.0" };

9

10 ORB orb;

11

12 Hashtable hash;

13

14 BillingServiceImpl(ORB orb) {

15 this.orb = orb;

16 hash = new Hashtable();

17 }

18

19 public void invoke(ServerRequest request) {

20 try {

21 System.out.println("DSI: invoke called, op = " + request.op\_name());

22 if (request.op\_name().equals("addNewCard") == true) {

23 /\*String personName, String card\*/

24 NVList nvlist = orb.create\_list(0);

25 Any any1 = orb.create\_any();

26 any1.insert\_string("");

27 nvlist.add\_value("arg1", any1, ARG\_IN.value);

28 Any any2 = orb.create\_any();

29 any2.insert\_string("");

30 nvlist.add\_value("arg2", any2, ARG\_IN.value);

31 // pass the NVList to the request to get values

32 request.params(nvlist);

33 addNewCard(nvlist.item(0).value().extract\_string(), nvlist.item(1).value().extract\_string());

34 TypeCode result\_tc = orb.get\_primitive\_tc(TCKind.tk\_void);

35 Any result\_any = orb.create\_any();

36 result\_any.type(result\_tc);

37 request.result(result\_any);

38 } else if (request.op\_name().equals("addMoney") == true) {

39 /\*String card, double money\*/

40 NVList nvlist = orb.create\_list(0);

41 Any any1 = orb.create\_any();

42 any1.insert\_string("");

43 nvlist.add\_value("arg1", any1, ARG\_IN.value);

44 Any any2 = orb.create\_any();

45 any2.insert\_double(0);

46 nvlist.add\_value("arg2", any2, ARG\_IN.value);

47 // pass the NVList to the request to get values

48 request.params(nvlist);

49 addMoney(nvlist.item(0).value().extract\_string(), nvlist.item(1).value().extract\_double());

50 TypeCode result\_tc = orb.get\_primitive\_tc(TCKind.tk\_void);

51 Any result\_any = orb.create\_any();

52 result\_any.type(result\_tc);

53 request.result(result\_any);

54 } else if (request.op\_name().equals("subMoney") == true) {

55 /\*String card, double money\*/

56 NVList nvlist = orb.create\_list(0);

57 Any any1 = orb.create\_any();

58 any1.insert\_string("");

59 nvlist.add\_value("arg1", any1, ARG\_IN.value);

60 Any any2 = orb.create\_any();

61 any2.insert\_double(0);

62 nvlist.add\_value("arg2", any2, ARG\_IN.value);

63 // pass the NVList to the request to get values

64 request.params(nvlist);

65 subMoney(nvlist.item(0).value().extract\_string(), nvlist.item(1).value().extract\_double());

66 TypeCode result\_tc = orb.get\_primitive\_tc(TCKind.tk\_void);

67 Any result\_any = orb.create\_any();

68 result\_any.type(result\_tc);

69 request.result(result\_any);

70 } else if (request.op\_name().equals("getCardBalance") == true) {

71 /\*String card, double money\*/

72 NVList nvlist = orb.create\_list(0);

73 Any any1 = orb.create\_any();

74 any1.insert\_string("");

75 nvlist.add\_value("arg1", any1, ARG\_IN.value);

76 // pass the NVList to the request to get values

77 request.params(nvlist);

78 double d = getCardBalance(nvlist.item(0).value().extract\_string());

79 TypeCode result\_tc = orb.get\_primitive\_tc(TCKind.tk\_double);

80 Any result\_any = orb.create\_any();

81 result\_any.insert\_double(d);

82 request.result(result\_any);

83 }

84 } catch (Exception ex) {

85 ex.printStackTrace();

86 System.out.println("DSIExample: Exception thrown: " + ex);

87 }

88 }

89

90 // implement an \_ids method to return repository ID of interface

91 public String[] \_ids() {

92 return myIDs;

93 }

94

95 public void addNewCard(String personName, String card) {

96 hash.put(card, new Double(0.0));

97 }

98

99 public void addMoney(String card, double money) {

100 synchronized (hash) {

101 Double d = (Double) hash.get(card);

102 if (d != null)

103 hash.put(card, new Double(d.doubleValue() + money));

104 }

105 }

106

107 public void subMoney(String card, double money) {

108 Double d = (Double) hash.get(card);

109

110 if (d != null)

111 hash.put(card, new Double(d.doubleValue() - money));

112 }

113

114 public double getCardBalance(String card) {

115 Double d = (Double) hash.get(card);

116 if (d != null)

117 return d.doubleValue();

118 else

119 return 0;

120 }

121 }

Листинг 9.1. Класс BillingServiceImpl

Итак, метод **invoke** вызывается брокером запросов при вызове любого метода объекта, и его единственным параметром является **ServerRequest** - объект, содержащий всю необходимую информацию о вызове: имя вызываемого метода, список параметров и их типы. Поэтому в первую очередь проверяется имя вызываемого метода - оно может быть получено с помощью вызова **op\_name()**,примененного к переданному **ServerRequest**.Далее, поскольку методы, как правило, принимают и возвращают аргументы, необходимо, во-первых, извлечь аргументы, переданные клиентом, а во-вторых, возвратить клиенту результат, если это необходимо.

Для извлечения из **ServerRequest** значений переданных клиентом аргументов необходимо выполнить следующие действия.

Создать список, в который затем будут считаны значения аргументов:

NVList nvlist = orb.create\_list(0);

Заполнить список объектами, в которые будут считаны значения переданных клиентом аргументов. В данном случае список заполняется двумя объектами типа **Any** (это специальный тип CORBA, совместимый со всеми типами), которым присваиваются некоторые начальные значения (пустые строки) и которые заносятся в список аргументов как принимаемые аргументы (направление передачи параметров - **in)**.

Any any1 = orb.create\_any();

any1.insert\_string("");

nvlist.add\_value("arg1", any1, ARG\_IN.value);

Any any2 = orb.create\_any();

any2.insert\_double(0);

nvlist.add\_value("arg2", any2, ARG\_IN.value);

Затем подготовленный список аргументов передается в ServerRequest и с помощью вызова request.params(nvlist) заполняется значениями, переданными клиентом.

Для того чтобы получить доступ к этим значениям, можно воспользоваться методом **extract\_<datatype>** (например, для получения значения первого переданного аргумента, при условии, что он является строковым, можно воспользоваться вызовом nvlist.item(0).value().extract\_string()).Следует обратить внимание, что для получения значения аргумента необходимо явно указывать его тип, вызывая соответствующий метод семейства **extract\***.

После того как все эти операции проделаны, имя вызываемого метода определено и переданные ему клиентом аргументы получены, серверный класс может их обработать. После обработки, в случае если вызываемый клиентом метод предполагает возврат результата, необходимо его сформировать:

TypeCode result\_tc = orb.get\_primitive\_tc(TCKind.tk\_double);

Any result\_any = orb.create\_any();

result\_any.insert\_double(d);

request.result(result\_any);

Сначала формируется **TypeCode** соответствующего типа (в данном случае предполагается, что метод возвращает результат типа **double)**.Затем создается объект для результата (типа **Any)**.Затем в этот объект помещается результат (с помощью одного из методов семейства **insert\*)**,после чего объект помещается в **ServerRequest** для возврата клиенту.

Следует обратить внимание, что даже если метод никакого результата не возвращает (имеет тип возврата **void)**,возврат все равно должен быть сформирован, только в качестве его типа должно быть указано **TCKind.tk\_void**.

**Класс BillingServiceServer**

Класс **BillingServiceServer** служит для создания экземпляра **BillingServiceImpl** и его регистрации в сервисе имен. Он ничем не отличается от подобных классов, рассмотренных нами ранее.

1 package com.asw.corba.ex3;

2

3 import org.omg.CosNaming.\*;

4 import org.omg.CORBA.\*;

5

6 public class BillingServiceServer {

7

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

9 try {

10 // create and initialize the ORB

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

12 // create servant and register it with the ORB

13 BillingServiceImpl BSRef = new BillingServiceImpl(orb);

14 orb.connect(BSRef);

15 System.out.println(BSRef);

16 org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");

17 NamingContext ncRef = NamingContextHelper.narrow(objRef);

18 // bind the Object Reference in Naming

19 NameComponent nc = new NameComponent("BillingService", "");

20 NameComponent path[] = { nc };

21 ncRef.rebind(path, BSRef);

22 java.lang.Object sync = new java.lang.Object();

23 synchronized (sync) { sync.wait(); }

24 } catch (Exception e) {

25 System.err.println("ERROR: " + e);

26 e.printStackTrace(System.out);

27 }

28 }

29 }

Листинг 9.2. Класс BillingServiceServer

**Класс BillingServiceClient**

Клиентский класс, так же как и серверный, не использует сгенерированные вспомогательные классы. Это означает, что необходимо вручную сформировать все вызовы к серверу, т.е. определить вызываемый метод, сформировать список передаваемых аргументов, осуществить вызов и получить результат.

1 package com.asw.corba.ex3;

2

3 import org.omg.CosNaming.\*;

4 import org.omg.CORBA.\*;

5

6 public class BillingServiceClient {

7 org.omg.CORBA.Object BSImpl;

8

9 ORB orb;

10

11 void testAddNewCard(String personName, String card) {

12 org.omg.CORBA.Request r2 = BSImpl.\_request("addNewCard");

13 r2.set\_return\_type(orb.get\_primitive\_tc(org.omg.CORBA.TCKind.tk\_void));

14 org.omg.CORBA.Any inc = r2.add\_in\_arg();

15 inc.insert\_string(personName);

16 org.omg.CORBA.Any inc2 = r2.add\_in\_arg();

17 inc2.insert\_string(card);

18 r2.invoke();

19 java.lang.Exception ex = r2.env().exception();

20 if (ex instanceof org.omg.CORBA.UnknownUserException) {

21 org.omg.CORBA.UnknownUserException userEx = (org.omg.CORBA.UnknownUserException) ex;

22 }

23 }

24

25 void testAddMoney(String card, double amount) {

26 org.omg.CORBA.Request r2 = BSImpl.\_request("addMoney");

27 r2.set\_return\_type(orb.get\_primitive\_tc(org.omg.CORBA.TCKind.tk\_void));

28 org.omg.CORBA.Any inc = r2.add\_in\_arg();

29 inc.insert\_string(card);

30 org.omg.CORBA.Any inc2 = r2.add\_in\_arg();

31 inc2.insert\_double(amount);

32 r2.invoke();

33 java.lang.Exception ex = r2.env().exception();

34 if (ex instanceof org.omg.CORBA.UnknownUserException) {

35 org.omg.CORBA.UnknownUserException userEx = (org.omg.CORBA.UnknownUserException) ex;

36 }

37 String result;

38 }

39

40 void testSubMoney(String card, double amount) {

41 org.omg.CORBA.Request r2 = BSImpl.\_request("subMoney");

42 r2.set\_return\_type(orb.get\_primitive\_tc(org.omg.CORBA.TCKind.tk\_void));

43 org.omg.CORBA.Any inc = r2.add\_in\_arg();

44 inc.insert\_string(card);

45 org.omg.CORBA.Any inc2 = r2.add\_in\_arg();

46 inc2.insert\_double(amount);

47 r2.invoke();

48 java.lang.Exception ex = r2.env().exception();

49 if (ex instanceof org.omg.CORBA.UnknownUserException) {

50 org.omg.CORBA.UnknownUserException userEx = (org.omg.CORBA.UnknownUserException) ex;

51 }

52 String result;

53 }

54

55 double testGetCardBalance(String card) {

56 org.omg.CORBA.Request r2 = BSImpl.\_request("getCardBalance");

57 r2.set\_return\_type(orb.get\_primitive\_tc (org.omg.CORBA.TCKind.tk\_double));

58 org.omg.CORBA.Any inc = r2.add\_in\_arg();

59 inc.insert\_string(card);

60 r2.invoke();

61 java.lang.Exception ex = r2.env().exception();

62 if (ex instanceof org.omg.CORBA.UnknownUserException) {

63 org.omg.CORBA.UnknownUserException userEx = (org.omg.CORBA.UnknownUserException) ex;

64 }

65 double result;

66 result = r2.return\_value().extract\_double();

67 System.out.println("Balance= " + result);

68 return result;

69 }

70

71 public void doTest(String args[]) throws Exception {

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

73 org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");

74 NamingContext ncRef = NamingContextHelper.narrow(objRef);

75 // resolve the Object Reference in Naming

76 NameComponent nc = new NameComponent("BillingService", "");

77 NameComponent path[] = { nc };

78 BSImpl = ncRef.resolve(path);

79 System.out.println(BSImpl);

80 this.testAddNewCard("Ivan", "1");

81 this.testAddMoney("1", 10.2);

82 this.testGetCardBalance("1");

83 this.testSubMoney("1", 8.1);

84 this.testGetCardBalance("1");

85 }

86

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

88 try {

89 BillingServiceClient bsc = new BillingServiceClient();

90 bsc.doTest(args);

91 } catch (Exception e) {

92 System.out.println("ERROR : " + e);

93 e.printStackTrace(System.out);

94 }

95 }

96 }

Листинг 9.3. Класс BillingServiceClient

Формирование динамического вызова метода сервера осуществляется следующим образом.

У полученной у сервиса имен объектной ссылки (она имеет тип **org.omg.CORBA.Object)** вызовом метода **request** (в качестве аргумента этому методу передается имя вызываемого метода сервера) получаем **org.omg.CORBA.Request** - объект, который используется для осуществления запросов к серверу.

Первое, что необходимо указать, - тип возвращаемого методом значения:

r2.set\_return\_type(orb.get\_primitive\_tc

(org.omg.CORBA.TCKind.tk\_double));

Затем необходимо поместить в запрос передаваемые аргументы:

Заorg.omg.CORBA.Any inc = r2.add\_in\_arg();

inc.insert\_string(card);

org.omg.CORBA.Any inc2 = r2.add\_in\_arg();

inc2.insert\_double(amount);

Для этого сначала объект нужно поместить в список передаваемых аргументов, а потом присвоить ему значение.

После того как тип возвращаемого значения определен, а список передаваемых методу аргументов сформирован, метод может быть вызван:

r2.invoke();

Для того чтобы определить, не произошло ли при обработке запроса сервером исключений, их (исключения) можно извлечь с помощью вызова:

java.lang.Exception ex = r2.env().exception();

Для извлечения результата, возвращенного серверным методом, снова нужно обратиться к **org.omg.CORBA.Request**.

double result = r2.return\_value().extract\_double();

Таким образом, без использования IDL и соответствующих вспомогательных классов, генерируемых на его основе, разработчикам приходится выполнять гораздо больше рутинной работы. Так, приходится вручную осуществлять диспетчеризацию вызовов в серверном коде, а также вручную обрабатывать передаваемые и возвращаемые параметры. Похожие задачи, по упаковке аргументов и распаковке результата, приходится выполнять и на клиенте. Однако применение рассмотренной техники программирования позволяет решать ряд задач, которые невозможно решить с помощью систем, ориентированных на работу с объектами, интерфейсы которых известны на момент компиляции.

**Компиляция и выполнение примера**

Компиляция приложений, использующих DII или DSI, ничем не отличается от компиляции ранее рассмотренных примеров.

Так же как для запуска других приложений CORBA, использующих сервис имен, требуется запуск этого сервиса:

tnameserv

Запуск компонентов приложения также не имеет особенностей:

G:\java com.asw.corba.ex3.BillingServiceServer

IOR:000000000000002249444c3a4a61766149444c2f44534942696c6c696e6753

6572766963653a312e3000000000000001000000000000006e000102000000000c

3139322e3136382e312e3100134d000000000019afabcb0000000002b42a5d5400

000008000000000000000014000000000000020000000100000020000000000001

000100000002050100010001002000010109000000010001010000000026000000

020002

DSI: invoke called, op = addNewCard

DSI: invoke called, op = addMoney

DSI: invoke called, op = getCardBalance

DSI: invoke called, op = subMoney

DSI: invoke called, op = getCardBalance

G:\java com.asw.corba.ex3.BillingServiceClient

IOR:000000000000002249444c3a4a61766149444c2f44534942696c6c696e6753

6572766963653a312e3000000000000001000000000000006e000102000000000c

3139322e3136382e312e3100134d000000000019afabcb0000000002b42a5d5400

000008000000000000000014000000000000020000000100000020000000000001

000100000002050100010001002000010109000000010001010000000026000000

020002

Balance= 10.2

Balance= 2.0999999999999996