**Лекция 4**

Технология, о которой мы говорили в предыдущем разделе, предполагает существенный объем программирования для реализации взаимодействия компонентов. Следующая рассматриваемая технология - RMI - позволяет существенно упростить разработку распределенных систем.

RMI дает возможность выполнять объекты Java на различных компьютерах или в отдельных процессах путем взаимодействия их друг с другом посредством удаленных вызовов методов. Технология RMI основана на более ранней подобной технологии удаленного вызова процедур ( RPC ) для процедурного программирования, разработанной в 80-х годах. RPC позволяет процедуре вызывать функцию на другом компьютере столь же легко, как если бы эта функция была частью программы, выполняющейся на том же компьютере. RPC выполняет всю работу по организации сетевых взаимодействий и маршалинга данных (т.е. пакетирования параметров функций и возврата значений для передачи их через сеть). Но RPC не подходит для передачи и возврата объектов Java, потому что она поддерживает ограниченный набор простых типов данных. Есть и другой недостаток у RPC - программисту необходимо знать специальный язык определения интерфейса ( IDL ) для описания функций, которые допускают удаленный вызов. Для устранения этих недостатков и была разработана технология RMI.

RMI представляет собой реализацию RPC на Java для распределенных коммуникационных взаимодействий "Java-объект - Java -объект". Объект Java регистрируется для удаленного доступа, что дает возможность клиентам получать удаленную ссылку на этот объект - она позволяет использовать этот объект дистанционно. Синтаксис вызова метода идентичен синтаксису вызова методов других объектов в той же программе. RMI обслуживает маршалинг данных через сеть и дает возможность программам на Java передавать законченные объекты Java с помощью механизма сериализации объектов Java. В составе J2SE имеются инструментальные средства создания требуемого кода для сетевых взаимодействий из определенных интерфейсов программы, это означает, что RMI не требует от программиста знания языка IDL. Кроме того, никакого нейтрального к языку IDL интерфейса не требуется, так как RMI поддерживает только Java ; достаточно собственных интерфейсов Java.

#### Создание распределенной системы с помощью RMI

В последующих разделах будет рассмотрен пример, использующий RMI. При этом предлагается несколько вариантов решения поставленной задачи.

В примере выполняются четыре основных действия:

* определение удаленного интерфейса с объявлениями методов, которые клиент может вызвать у удаленного объекта;
* определение реализации удаленного объекта для удаленного интерфейса;
* определение клиентского приложения, которое использует удаленную ссылку, чтобы взаимодействовать с реализацией интерфейса;
* компиляция и выполнение удаленного объекта и клиента.

#### Первый пример

Решать предложенную задачу мы будем поэтапно - постепенно наращивая сложность наших программ, при этом иллюстрируя те или иные особенности применяемых технологий.

Для начала рассмотрим совсем простую задачу: создадим систему, функционально состоящую из двух компонентов - сервера (процессингового центра) и клиента (касса). Будем предполагать, что сервер в системе один (именно он обладает всей информацией о зарегистрированных картах и их балансах), а касс много и на них проходят операции регистрации новых карт, а также операции изменения баланса карт (соответственно - оплаты и занесения наличных).

На данном этапе мы никак не будем учитывать то, что связь между нашими объектами неустойчивая, - более того, мы будем считать ее устойчивой. Кроме того, в целях сокращения размеров примеров не будут рассматриваться вопросы долговременного хранения обрабатываемой информации - при перезапуске наша система будет "забывать" все занесенные в нее карты и их балансы.

**Определение удаленного интерфейса**

При проектировании сервера нам необходимо определить выполняемые им функции и, соответственно, методы, которые будет вызывать клиент. Поскольку операциями нашей системы являются регистрация новой карты, занесение денег на карту, оплата картой покупки и просмотр баланса карты, будем считать, что наш сервер будет обладать четырьмя перечисленными методами - именно их клиент (касса) и будет вызывать. Поскольку единственным, что отличает одну карту от другой, является ее номер (код) - он и будет служить идентификатором, карты и поэтому будет присутствовать во всех методах сервера в качестве параметра. Поскольку этот код, вообще говоря, алфавитно-цифровой, будем использовать тип данных String для его хранения.

Удаленные методы, посредством которых клиент взаимодействует с удаленным объектом, используя RMI, должны быть определены в удаленном интерфейсе. Соответственно, первый этап при создании распределенного приложения с помощью RMI состоит в определении удаленного интерфейса, который описывает эти удаленные методы. Чтобы создать удаленный интерфейс, необходимо определить интерфейс, который будет расширять интерфейс **java.rmi.Remote**.Интерфейс **Remote** представляет собой тегирующий интерфейс - он не объявляет каких-либо методов, поэтому не обременен реализацией класса. Распределенное RMI -приложение должно экспортировать объект класса, который реализует интерфейс **Remote**,чтобы сделать этот удаленный объект доступным для приема удаленных вызовов метода из любой виртуальной машины Java, которая имеет соединение с компьютером, где выполняется удаленный объект.

Интерфейс **BillingService** ([пример 5.1](https://www.intuit.ru/studies/courses/633/489/lecture/11079?page=1#example.5.1)), который расширяет интерфейс **Remote** (строка 9), представляет собой интерфейс для нашего удаленного объекта (сервера). В строках 10-17 объявляются методы для работы с пластиковыми картами. Удаленный объект должен реализовать все объявленные в удаленном интерфейсе методы.

1 // BillingService.java

2 // Интерфейс BillingService объявляет методы для работы

3 // с пластиковыми картами

4 package com.asw.rmi.ex1;

5

6 // Набор базовых пакетов Java

7 import java.rmi.\*;

8

9 public interface BillingService extends Remote {

10 // определение новой карты

11 public void addNewCard(String personName, String card) throws RemoteException;

12 // добавить денежные средства на карту

13 public void addMoney(String card, double money) throws RemoteException;

14 // снять денежные средства с карты

15 public void subMoney(String card, double money) throws RemoteException;

16 // получение баланса карты

17 public double getCardBalance(String card) throws RemoteException;

18 }

Листинг 5.1. Интерфейс BillingService

Когда узлы взаимодействуют между собой по сети, есть вероятность возникновения проблем при таких взаимодействиях. Например, компьютер сервера может выйти из строя или может отказать какой-либо сетевой ресурс. Поэтому для контроля подобных проблем взаимодействия в сети каждый метод в интерфейсе **Remote** должен содержать throws для указания, что метод может возбуждать контролируемые исключения **RemoteException**.

RMI использует механизм сериализации по умолчанию Java для передачи параметров методу и возврата значений через сеть. В связи с этим все параметры метода и возвращаемые значения должны иметь описатель **Serializable** или один из примитивных типов.