Теоретический материал для индивидуального задания №10

Вызов удаленных методов (RMI)

Технологии вызова удаленных методов реализованы в пакетах `java.rmi` и `java.rmi.server`. Вызов удаленных методов является мощной технологией для разработки сетевых приложений, освобождающей программиста от необходимости заботиться о деталях реализации сетевых соединений на нижнем уровне.

В этой модели сервер определяет объекты, которые могут использоваться удаленными клиентами. Клиенты вызывают методы удаленных объектов так же, как если бы они были локальными объектами, выполняющимися внутри той же виртуальной машины, что и клиент. Технология RMI скрывает лежащий в ее основе механизм транспортировки параметров методов и возвращаемых значений через сеть. Параметр и возвращаемое значение могут быть либо значением примитивного типа, либо любым сериализуемым объектом.

Для того чтобы создать приложение на базе RMI:

1) Создайте интерфейс, расширяющий `java.rmi.Remote`. В этом интерфейсе определены экспортируемые методы, реализуемые удалёнными объектами (то есть методы, реализуемые сервером и вызываемые удаленным клиентом). Каждый метод этого интерфейса должен быть объявлен как генерирующий исключение `java.rmi.ReraoteException`, которое является базовым классом других классов исключений RMI. Каждый удаленный метод должен объявлять, что он может сгенерировать `RemoteException` из-за того, что существуют ситуации, приводящие к возникновению ошибок во время процесса вызова удаленных методов через сеть.

2) Определите класс, производный от `java.rmi.server.UnicastRemoteObject` (или от потомка), реализующий ваш удаленный интерфейс. Этот класс представляет удаленный, или серверный, объект. Кроме объявления того, что его удаленные методы генерируют исключения `RemoteException`, удаленный объект не должен делать что-либо особенное, чтобы позволить вызывать его методы удаленно. Объект `UnicastRemoteQbject` (и вся остальная часть инфраструктуры RMI) обрабатывают это автоматически.

3) Напишите программу (сервер), создающую экземпляр вашего удаленного объекта. Экспортируйте объект, сделав его доступным для использования клиентами путем регистрации имени объекта в службе реестра. Как правило, это выполняется с помощью класса `java.rmi.Naming` и программы `rmiregistry`. Кроме того, серверная программа может сама выполнять функции сервера реестра, используя класс `LocateRegistry` и интерфейс `Registry` из пакета `java.rmi.registry`.

4) При использовании RMI клиент и сервер не взаимодействуют непосредственно. На стороне клиента ссылка на удаленный объект реализуется в виде экземпляра класса заглушки. Когда клиент вызывает удаленный метод, в действительности вызывается метод объекта-заглушки. Заглушка производит необходимые сетевые операции по передаче этого вызова находящемуся на сервере классу каркаса. Этот каркас транслирует пришедший по сети запрос в вызов метода

серверного объекта, а затем передает возвращенное им значение обратно заглушке, которая, в свою очередь, возвращает его клиенту. Все это представляет собой довольно сложную систему, но прикладным программистам никогда не приходится думать о заглушках и каркасах; они генерируются автоматически утилитой `rmic`. До версии Java 5 заглушки приходилось создавать вручную, а в современных версиях Java эта стадия уже необязательна (см. документацию на `java.rmi.server.UnicastRemoteObject`). Процесс создания заглушки и каркаса вручную описан далее.

После того как вы откомпилируете серверную программу, воспользуйтесь утилитой `rmic`, чтобы сгенерировать для удаленного объекта заглушку (`stub`) и каркас (`skeleton`).

Вызовите из командной строки `rmic`, указав имя удаленного класса объекта (не интерфейса). Она создаст и откомпилирует два новых класса с суффиксами `_Stub` и `_Skel`. Если сервер использует службу реестра по умолчанию, предоставленную классом `Naming`, вы должны запустить сервер реестра, если он еще не запущен путем вызова программы `rmiregistry`.

5) Сейчас вы можете написать клиентскую программу, использующую экспортированный сервером удаленный объект. Прежде всего, клиенту необходимо получить ссылку на удаленный объект, используя класс `Naming` для поиска объекта по имени. Это имя обычно задается в форме `rmi:URL`. Удаленная ссылка, которая будет возвращена, представляет собой экземпляр интерфейса `Remote` объекта (или, более точно, объект-заглушку удаленного объекта). Как только клиент получил этот удаленный объект, он может вызывать его методы точно так же, как он вызывал бы методы локального объекта. Он должен только знать, что все удаленные методы могут генерировать объекты исключений `RemoteException` и что при возникновении сетевых ошибок это может случаться в самый неожиданный момент.

6) Наконец, запустите и серверную программу, и клиент

Многопользовательская область

Рассмотрим пример реализации многопользовательской игровой среды средствами RMI.

Многопользовательская область, или `MUD` (`multiuser domain`), представляет собой программу (сервер), дающую многим людям (клиентам) возможность взаимодействовать друг с другом и с общим виртуальным окружением. Окружением обычно является ряд комнат, или мест, связанных друг с другом разнообразными выходами (`exit`). Каждая комната, или место, имеет текстовое описание, служащее в качестве декорации и задающее тон взаимодействия между пользователями. Многие из ранних реализаций `MUD` были составлены из темниц с описаниями комнат, отражающими темную, подземную натуру этого воображаемого мира. Фактически сокращение `MUD` первоначально означало «многопользовательская темница» (`multiuser dungeon`). Некоторые `MUD` служат, прежде всего, в качестве чатов для их клиентов, тогда как другие больше похожи на некую разновидность старых приключенческих игр, которые сконцентрированы на исследовании окружения и решении головоломок. Другие являются упражнениями в творчестве и групповой динамике, разрешая пользователям добавлять в `MUD` новые комнаты и элементы.

В примере приведены классы и интерфейсы, которые определяют простую расширяемую пользователем систему MUD. Такая программа, как этот пример, демонстрирует, насколько парадигма RMI-программирования расширяет модель клиент-сервер. Как мы увидим дальше, MudServer и MudPlace являются серверными объектами, создающими окружение MUD, внутри которого взаимодействуют пользователи. Но в то же время каждый пользователь, находящийся внутри MUD, представлен удаленным объектом MudPerson, который при взаимодействии с другими пользователями действует в качестве сервера. Вместо использования единственного сервера и набора клиентов, эта система действительно является распределенной сетью удаленных объектов, где все взаимодействуют друг с другом. Кто из объектов в действительности является сервером, а кто - клиентом, зависит от вашей точки зрения.

Для того чтобы понять систему MUD, следует дать краткий обзор ее архитектуры. Класс MudServer - это простой удаленный объект (и самостоятельная серверная программа), который является точкой входа в MUD и отслеживает имена всех комнат в пределах MUD. Несмотря на свое имя, объект MudServer не предоставляет услуг, о которых большинство пользователей думает как о MUD. Это работа класса MudPlace.

Каждый объект MudPlace представляет собой отдельную комнату внутри MUD. У каждой комнаты есть имя, описание, список находящихся в ней предметов и людей (пользователей), выходы из этой комнаты и другие комнаты, в которые эти выходы ведут. Выход может вести к смежному объекту MudPlace, находящемуся на этом же сервере, или к объекту MudPlace другой MUD, расположенной вовсе на другом сервере. Таким образом, MUD-окружение, с которым взаимодействует пользователь, в действительности является сетью объектов MudPlace. Описания комнат и предметов и сложность связей между комнатами дают среде MUD то богатство, которое делает ее интересной пользователю.

Каждый пользователь, или человек, в системе MUD представлен объектом MudPerson. MudPerson - это удаленный объект, имеющий два метода. Один из них возвращает описание человека (то есть то, что видят другие люди, когда они смотрят на этого человека), а второй доставляет человеку (или пользователю, которого представляет MudPerson) некоторое сообщение. Эти методы позволяют пользователям смотреть друг на друга и говорить друг с другом. Когда два пользователя встречаются друг с другом в данном MudPlace и начинают беседовать, MudPlace и сервер, на котором работает MUD, больше не имеют значения. Благодаря RMI два объекта MudPerson могут связываться друг с другом напрямую.

Удаленные интерфейсы MUD

Класс Mud, играющий роль контейнера внутренних классов и интерфейсов (и одной константы), используемых в оставшейся части MUD-системы. Наиболее важно то, что в Mud определено три удаленных интерфейса: RemoteMudServer, RemoteMudPerson и RemoteMudPlace. Они определяют удаленные методы, которые реализуются объектами MudServer, MudPerson и MudPlace соответственно.

Сервер MUD

Класс MudServer является автономной программой, которая начинает выполнение MUD. Также она предоставляет реализацию интерфейса RemoteMudServer. Как было

отмечено выше, объект MudServer служит просто входом для MUD: он - не сама MUD. Поэтому он является довольно простым классом. Одна из его наиболее интересных особенностей - это использование классов сериализации java.io и классов архивирования java.util.zip для сохранения состояния MUD с целью его восстановления позже.

Класс MudPlace

Класс MudPlace реализует интерфейс RemoteMudPlace и действует в качестве сервера для отдельной комнаты или места, находящегося внутри MUD. Это именно тот класс, который хранит описание комнаты и содержит списки находящихся в ней людей и предметов, а также выходов из нее. Это достаточно большой класс, но многие из определенных в нем удаленных методов имеют простые, а зачастую даже тривиальные реализации. Методы go(), createPlace() и linkTo() относятся к более сложным и интересным методам, они управляют сетевым обменом между объектами MudPlace.

Обратите внимание, что класс MudPlace - сериализуемый, поэтому MudPlace (и все связанные с ним комнаты) могут быть сериализованы вместе с MudServer, который ссылается на них. Однако поля names и people объявлены как transient, то есть они не сериализуются вместе с комнатой.

Класс MudPerson

Класс MudPerson является самым простым из всех удаленных объектов в системе MUD. Он реализует два удаленных метода, определенных в интерфейсе RemoteMudPerson, и кроме этого определяет несколько не удаленных методов, используемых классом MudClient. Удаленные методы довольно просты: один из них просто возвращает вызывающему объекту строку описания, а другой записывает сообщение в поток, где его может увидеть пользователь.

Клиент MUD

Класс MudClient является клиентской программой для системы MUD. Он использует метод Naming.lookup() для поиска объекта RemoteMudServer, который представляет указанную MUD на заданном хосте. Затем программа вызывает метод getEntrance() иди getNamedPlace() этого объекта для получения начального объекта MudPlace и введения в него пользователя. После этого программа запрашивает у пользователя имя и описание для объекта MudPerson, который будет представлять его в системе MUD, создает объект MudPerson с этим именем и описанием, а затем помещает его в начальный объект RemoteMudPlace. Наконец, программа входит в цикл, в котором просит пользователя ввести команду и обрабатывает ее. Большинство команд, которые поддерживает этот клиент, просто вызывают один из удаленных методов объекта RemoteMudPlace, представляющего текущее положение пользователя в MUD. Конец командного цикла состоит из ряда секций catch, которые обрабатывают большое количество ошибочных ситуаций.

Перед началом использования класса MudClient вы должны запустить MudServer. Запустив сервер, вы можете запустить клиент.

Демонстрационные примеры (в папке Примеры)

В примере 1 показан пример клиент-сервер банковской системы.
В примере 2 показан пример клиент-сервер системы MUD.

Дополнительный материал смотрите:
в презентации - `Java-УП-10.ppt`
в главе 3 книги _Books\corejava2_2.djvu

Литература

1. Хабибуллин И. Ш. Java 7. — СПб.: БХВ-Питербург, 2012
2. Г. Шилдт. Java . Полное руководство, 8-е издание, М.: ООО “И.Д. Вильямс”,2012
3. Кей С. Хорстман. Java2 Основы. Том 1. С.-Питербург. 2007 (_Books\ corejava2_1.djvu)
4. Кей С. Хорстман. Java2 Тонкости программирования. Том 2. С.-Питербург. 2007 (_Books\ corejava2_2.djvu)
5. http://docs.oracle.com/javase/8/docs/