Алгоритм

Технологія RMI забезпечує віддалений виклик методів об’єктів в розподілених системах і може використовуватися для організації синхронної клієнт-серверної взаємодії застосунків. В рамках технології RMI додаток-сервер створює об’єкт і реєструє його для віддаленого доступу. Клієнтські програми отримують посилання на віддалений об’єкт і працюють з ним таким чином, як ніби він є для них звичайним локальним об’єктом. Звернення до серверного об’єкту клієнт здійснює з використанням інтерфейсу об'єкта. Технологія RMI приховує від розробників застосунків реалізацію процесу мережевої взаємодії між клієнтським і серверним застосунками. За формування мережевих пакетів і їх передачу повністю відповідає проміжне ПО RMI. Клієнт, викликаючи через інтерфейс метод віддаленого об'єкта, насправді звертається до спеціалізованого локального об'єкту-заглушці – стаб (stub). Стаб упаковує параметри виклику методу в байтовое повідомлення. При цьому параметри-об'єкти піддаються серіалізації. Процес пакування параметрів називається маршалінг (marshaling). Сформоване повідомлення передається по мережі на сервер. Його отримує скелетон (skeleton) – серверна заглушка. Завдання скелетона – витягти параметри з отриманого повідомлення і викликати необхідний метод серверного об'єкта (який, зауважимо, є локальним для скелетона). Результат виклику методу скелетон упаковує в байтове повідомлення і передає назад стаб клієнта. Стаб клієнта розпаковує повідомлення і повертає результат викликавшій його програмі. За стабом і скелетоном ховаються два допоміжних рівня: рівень віддалених посилань і транспортний рівень. Рівень віддалених посилань реалізує протокол взаємодії, незалежний від стабу і скелетона. Даний рівень знає, як потрібно керувати посиланнями клієнта на віддалені об'єкти, і яку модель передачі даних використовувати при взаємодії клієнта і сервера. Транспортний рівень відповідає за доставку мережевих пакетів між клієнтом і сервером.

Зв'язування клієнта і сервера здійснюється через спеціальну службу – службу імен та каталогів. Служба імен або каталогів зазвичай виконується на відомому всім клієнтам хості і має відомий номер порту. RMI може використовувати багато різних служб каталогів, включаючи Java Naming and Directory Interface (JNDI). У той же час, RMI включає в себе просту стандартну службу, звану реєстром RMI (rmiregistry). Реєстр RMI працює на серверній машині, яка містить об’єкти віддалених служб. За замовчуванням, реєстр використовує порт 1099. Основна функція реєстру RMI – надання клієнтам посилань на віддалені серверні об’єкти. Сервер, який бажає надати свій об’єкт віддаленим застосункам, попередньо реєструє його в реєстрі RMI під деяким загальновідомим ім’ям. Клієнти, що бажають працювати з віддаленим об’єктом, звертаються до реєстру RMI і отримують посилання на об’єкт по його імені.

Розробка клієнт-серверної системи на базі RMI в цій лабораторній роботі здійснювалась у такій послідовності:

1. Розробка інтерфейсу для віддаленого (серверного) об’єкта (інтерфейс надалі повинен бути доступний як для клієнтського, так і серверного застосунків).
2. Розробка класу серверного об’єкта, що підтримує даний інтерфейс.
3. Розробка серверного застосунку (застосунку, що керує серверним об’єктом).
4. Розробка клієнтської програми (застосунку, що звертається до віддаленого серверного об’єкту).

При роботі клієнт-серверної системи повинен дотримувався наступний порядок дій:

1. Запускалася служба реєстру RMI (це було зроблено в самому серверному застосунку).
2. Створювався серверний об’єкт і реєструвався в реєстрі RMI.
3. Запускалися клієнтські програми, отримували з реєстру RMI посилання на віддалений об’єкт, і викликали методи об’єкта.

Методи серверного об’єкта, які планувалося надавати віддаленим клієнтам через RMI, бути описані за допомогою інтерфейсу віддаленого об’єкта. Такий інтерфейс розширював базовий інтерфейс java.rmi.Remote. Крім того, всі методи інтерфейсу були оголошені з можливістю генерації спеціального виключення RemoteException. Даний виняток сигналізував про проблеми мережевої взаємодії при виклику віддаленого методу. Серверний об’єкт реалізовував описаний вище інтерфейс. При цьому клас серверного об’єкта успадковувався від класу UnicastRemoteObject. Клас UnicastRemoteObject (пакет java.rmi.server) представляє базові функціональні можливості, які необхідні віддаленим об’єктам для обслуговування віддалених запитів. Конструктори і методи класу UnicastRemoteObject збуджують контрольований виняток RemoteException, тому спадкоємці класу UnicastRemoteObject визначали конструктори, які також збуджують виняток RemoteException. Застосунок-сервер створював серверний об'єкт і реєстрував його в реєстрі RMI під певним ім’ям. При цьому створювався локальний реєстр безпосередньо з самого серверного додатка. Статичний метод createRegistry класу java.rmi.registry.LocateRegistry запускав службу реєстру RMI і повертав посилання на неї. Метод приймає як параметр номер порту TCP, який буде прослуховувати служба. Для роботи з реєстром використовувався інтерфейс java.rmi.registry.Registry. Для реєстрації об’єкта в реєстрі використовувався метод rebind інтерфейсу Registry. В даному випадку, замість повного URL вказувалося тільки ім’я, під яким об’єкт був зареєстрований.

Клієнтська програма звертається до віддаленого реєстру RMI і отримує посилання на серверний об’єкт. Для роботи з об’єктом (для виклику віддалених методів) клієнтський застосунок використовує інтерфейс серверного об’єкта. Метод lookup класу Naming повертав посилання на серверний об’єкт, зареєстрований під потрібним ім’ям в віддаленому реєстрі RMI, служба якого виконувалася за адресою 127.0.0.1 (localhost) і прослуховувала порт за замовчуванням (1099). Для роботи з серверним об’єктом клієнт використовував інтерфейс RMIServer.

При розробці ПО для лабораторної роботи №3 було звернено увагу на наступні особливості RMI:

1. Передані в віддалений метод параметри, а також значення, що поверталися могли відноситися до примітивних типів java або бути локальними і віддаленими об’єктами. Передача локальних об’єктів здійснювалася «за значенням». При передачі таких об’єктів RMI використовувала механізм серіалізації. У зв’язку з цим класи переданих локальних об’єктів підтримували інтерфейс java.io.Serializable.
2. При виклику клієнтом методу віддаленого об’єкта виконуючою частиною системи на стороні сервера формувався (або вибирався з числа вже створених) потік, в якому і відбувався виклик методу. Таким чином, ставало можливим одночасне виконання методів сервера декількома клієнтами. Тому, була необхідність синхронізувати доступ до полів класу з віддалених методів.