RMI

Az RMI (Remote Method Invocation), azaz távoli metódushívás egy olyan eszköz a Java nyelvben, mely lehetővé teszi más VM-ben (Virtual Machine – virtuális gép) elhelyezkedő objektumok metódusainak meghívását. A rendszert úgy tervezték, hogy a hálózaton elosztott objektumok viselkedése hasonló legyen a helyi objektumok viselkedésével, de ne egyezzen meg, a teljes transzparencia biztosítása nem volt cél. A rendszer elrejti a kommunikációt végző részt, és a metódushívás hasonló a helyi, azaz lokális objektumok metódusainak meghívásával, bizonyos megszorításokkal és kötöttségekkel. Ebben a fejezetben szeretnék kitérni az RMI rendszer működésére, mely szorosan kapcsolódik a munkámhoz, illetve azon különbségekre, melyeket figyelembe kell venni a használat és az implementáció során.

Mivel a Java nyelv egy objektumorientált programozási nyelv, ezért egy objektumorientált tervezési módszert követel meg egy rendszer fejlesztésekor. Szükséges volt tehát egy olyan eszközrendszer, mely illeszkedik az objektumorientált tervezési módszerhez, és segítségével az alkalmazást olyan részekre, objektumokra lehessen bontani, úgy hogy heterogén rendszereken és számítógépes hálózatokra telepítve is működőképes legyen. A Java nyelvben két ilyen eszköz is található, mégpedig a CORBA felület, mely programozási nyelvtől független megoldást kínál erre a problémára, illetve a távoli metódushívás, mely feltételezi, hogy az objektumok mind egy Java virtuális gépen helyezkednek el, és a hálózaton keresztül kommunikálhatnak. Ezek közül én az utóbbit választottam, mivel a Java környezet elég sok platformon elérhető (Solaris SPARC/x86, Linux x86, Microsoft Windows), illetve nincs olyan kész rendszer más programozási nyelven megírva, melyhez a rendszeremnek illeszkednie kellene.

Az RMI másik hatalmas előnye a Java-ban alkalmazott objektummodellhez való illeszkedésen kívül a kommunikáció egyszerűsége. Ugyanis a széles körben alkalmazott kliens-szerver modellben a rendszer részei közötti adatátvitel az I/O köré épül, sokszor saját protokollt kell kiépíteni az információcserére, illetve szinkronizációra. Ilyenkor a program írójának minden kapcsolódási pontnál implementálni kell a hálózati kommunikációt megvalósító eljárásokat, az adatátvitelt és szinkronizációt. (Ez annyit jelent, hogy létre kell hozni egy csatornát a kommunikáló felek között, ami általában egy TCP csatorna, szinkronizálni, kódolni,

illetve dekódolni az átvitelre váró adatokat.) Ezen eljárások hasonlítanak egymásra, de nehéz általános megoldást találni megvalósításukra a speciális esetek miatt, illetve könnyű hibát ejteni. Ezzel szemben az RMI egy magasabb szintű absztakciós eszköz, melynél az információáramlás a metódusok hívásakor átadott paraméterekkel oldható meg. Nagyon fontos, hogy a paraméterek csakis szerializálható adattípusok lehetnek.

Serialization

Egy objektum szerializáció során egy bájtfolyammá alakul, mely ezek után használható a Java nyelv szabványos kimenet és bemenet kezelési módszereivel. Ennek ellenkezője a deszerializáció, mely a visszaalakítást jelenti a bájtsorozatból objektummá. A szerializáció szó, mint technika takarhatja mind az előbbi értelemben vett szerializációt, mind a deszerializációt. A szerializáció alkalmas például egy objektum háttértárra való kiírására, azaz állományba mentéshez, illetve alkalmas a VM-ek közötti kommunikáció megvalósítására, ugyanis a szerializált objektum a hálózaton átküldhető, mint bájtfolyam, és a másik oldalon visszaalakítható objektummá.

A szerializáció egy nagyon előnyös tulajdonsága, hogy egy objektum szerializálásakor nem csak az objektum kerül szerializálásra, hanem minden olyan objektum is, amire az eredeti objektum valamely mezője referenciát tartalmaz. És ez ismétlődik ezen objektumokra rekurzívan.

RMI működése

Ebben a fejezetben az RMI alapvető működésére szeretnék kitérni, nem említve az implementációs kérdéseket, amelyekre később fogok kitérni, közvetlenül a rendszerhez kapcsolódó problémák kapcsán.

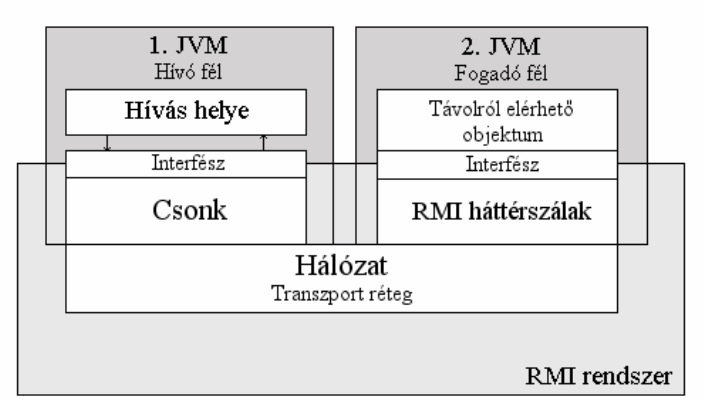
Mint említettem, az RMI olyan eszközt ad a kezünkbe, mellyel más virtuális gépeken lévő objektumok metódusait tudjuk meghívni. Így a rendszer alapfogalma a távoli objektum. Viszont egy objektum nem minden metódusa hívható távolról (, csupán azok, melyek egy úgynevezett távoli interfészben definiálva vannak), és fontos momentum, hogy nem biztosítja az osztályok távoli elérését (,ami azt jelenti, hogy egy osztály konstruktor műveletei nem lesznek használhatóak).

Ezen objektumok használata teljesen megegyezik a távolról nem hívható objektumok használatával, azaz a programozónak csak nagyon kevés helyen kell ezzel foglalkoznia, azaz a rendszer teljesen transzparens módon viselkedik.

Az RMI rendszer nem más, mint Java osztályok gyűjteménye, azaz nem nyelvi elem. Mivel a hálózati kommunikációt végző rész teljesen láthatatlan, ezért mégis annak tűnik. Ebből következik, hogy egy saját RMI rendszer implementálása Java nyelven lehetséges, a felmerülő problémák és azok megoldásainak ismeretében.

Fontos megjegyezni, hogy a Java fejlesztőkörnyezettel adott RMI rendszer saját szemétgyűjtő mechanizmussal is rendelkezik, és mivel működése hasonló a Java nyelvben, illetve VM-ben implementálttal, így erre nyugodtan hagyatkozhatunk, néhány megkötéssel.

A Java nyelvben nincsen olyan referencia, mely egy másik virtuális gépen elhelyezkedő objektumot címezne meg, ezért ehelyett egy úgynevezett csonkot kell használni. A csonk egy olyan objektum, mely a lokális virtuális gépen helyezkedik el, és hasonlít a távoli objektumra azzal a különbséggel, hogy nincs benne implementálva a távoli objektum metódusainak törzse. A metódushívást tehát úgy kell elképzelni, hogy a csonk egy metódusát hívjuk meg, mely a JRMP (Java Remote Method Protocol) protokollt használva továbbítja a hívást a távoli objektumnak, szerializálva a paramétereket, blokkolva a végrehajtást, majd ha visszaérkezik a visszatérési érték a hálózaton, deszerializálja azt, és visszaadja azt a hívó félnek. A csonk elkészítése automatikus, a Java fejlesztőkörnyezetben találunk hozzá eszközt. Tehát a csonk használata teljesen transzparens, egyenértékű a távoli referenciával. A hívás során a hálózaton a következő információk mennek át, melyek el vannak rejtve: a hívott objektum azonosításához szükséges információk, a hívott metódus azonosításához szükséges információk, illetve a metódushívás aktuális paraméterei, szerializálva. Fontos, hogy más-más virtuális gépen az ugyanarra az objektumhoz tartozó csonk igaz ugyan, hogy különböző objektum, de ugyanarra az objektumra vonatkozó referencia.



Míg a fogadó oldalon RMI háttérszálak futnak, melyek várják a beérkező távoli metódushívásokat. Amint kapnak egy ilyent, továbbadják azt a megcímzett objektumnak, mely ugyanazon a virtuális gépen helyezkedik el. Ebben már implementálva van a metódus törzse, így végrehajtódik az. A visszatérési értéket aztán visszaküldi a hívó oldalon szereplő csonknak, mely továbbadja a hívónak. Az előlünk elrejtett információk, melyek átmennek a hálózaton: a távoli metódus visszatérési értéke, és a kivétel, melyet vagy a távoli objektum dobott, vagy az RMI rendszer működése közben lépett fel. A kivételkezelés elosztott környezetben bonyolult probléma, megoldását később ismertetem, az implementációs részben. Annyit azonban megjegyzek, hogy a kivételeket mindig a hívó oldalon kell lekezelni, amik szintén szerializálva mennek át a hálózaton. Az RMI rendszer működése közben fellépett hibák válthatják ki a távoli metódushívás kivételeit, melyet például a hálózat megszakadása vagy a hívott fél helytelen működése okozhat. Egyéb dobott kivételeknél a szerializáció során elveszhetnek egyéb járulékos információk, melyek a hibakeresést jelentősen megkönnyítenék.

Ebből is látszik, hogy a paraméterként átadott objektumok nem mennek vissza a hívó félhez. Ez azt jelenti, hogy a paramétereken véghezvitt változás nem aktualizálódik a hívó oldalon, azaz érték szerinti paraméterátadásról beszélünk. Ez közvetlenül abból következik, hogy a deszerializáció során a hívott oldalon egy másolat keletkezik az eredeti objektumból.

Exportálás

Abban az esetben, mikor példányosítunk egy objektumot, ami távolról elérhetővé akarunk tenni, közölni kell az RMI háttérszálakkal a készen állását, hogy most már fogadhat távoli metódushívásokat erre az objektumra vonatkozólag, illetve

azért, hogy tudja, hova kell a hívást továbbadni. A közlést exportálásnak nevezzük. Természetesen egy objektumot unexportálni is lehet, ezek után nem érhető el többet RMI-n keresztül. Exportáláskor létrejön egy csonk példány is a hívott oldalon, melyet használva a hívó oldalon érhetjük el az eredeti, távolról elérhető objektumot.

RMI registry

Ahhoz, hogy referenciához jussunk a távolról hívható objektumra, szükségünk van a hozzá tartozó csonkra. De ahogy az előző fejezetben említettem, a csonk példány a hívott oldalon keletkezik, tehát valahogy el kell jutnia a hívó félhez.

Ennek lekérése az RMI registry-n keresztül történik, méghozzá RMI technikával, ugyanis a registry nem más, mint egy távolról elérhető objektum, melynek távolról hívható metódusaival rendelhetünk egy objektumhoz egy referenciát, kérhetjük le azt, illetve távolíthatjuk el a registry-ből.

A registry csonk és név párokat tartalmaz. Egy távoli objektum azonosítását egy speciális cím teszi lehetővé, melyben szerepel a számítógép neve, opcionálisan egy port, illetve az objektum neve.

A Java fejlesztő környezetben kapunk egy segédprogramot, mely létrehoz egy registry példányt, és exportálja azt. Erre azért van szükség, mivel a registry is egy távolról elérhető objektum. Viszont honnan tud a hívó fél referenciát szerezni a registry-re? Mivel tudjuk ennek a címét és a port számát, valamint speciális azonosítóval rendelkezik, ezért csonkot generálhatunk anélkül, hogy kapcsolatba lépnénk az registry-t példányosító és azt exportáló virtuális géppel. Persze saját magunk is implementálhatunk RMI registry-t, sőt mi is példányosíthatjuk és exportálhatjuk a programból. Én az utóbbi megoldást választottam, ahogy azt az implementációs részben fogom kifejteni, ugyanis a hívott fél és a registry ugyanazon a host-on, ugyanazon virtuális gépen szerepel, azaz a szerverem tartalmazza a registry-t is.

Fontos megjegyezni, hogy nem csak az registry-n keresztül kaphatunk meg egy referenciát ez távolról elérhető objektumra, hanem egy távoli metódushívás visszatérési értékeként is megkaphatunk egy hivatkozást, ami nem más mint egy csonk példány.

Fontos kérdések még az RMI-vel kapcsolatban a biztonsági feltételek, lévén szó hálózati kommunikációról, illetve az elosztott szemétgyűjtés megvalósítása. Mivel ezen problémák rendszerem fejlesztése során nem nagyon jöttek elő, ezekre pontosabban kitérni nem fogok. Továbbfejlesztési javaslatoknál említeni fogom a rendszer biztonságosabbá tételét. A szemétgyűjtést viszont teljesen úgy kezeltem, mintha nem elosztott környezetről lenne szó, ezen használat helyességének ellenőrzése is várat még magára.