RMI-IIOP – кратък технически обзор

Светлин Наков
www.nakov.com

Въведение

RMI-IIOP възниква като отговор на естествената нужда от взаимодействие между Java RMI-базираните и CORBA-базираните разпределени приложения. Тази нуждата от взаимодействие между двата стандарта е осъзната още през 1999 г. от привържениците на двете заинтересовани страни – Java общността и CORBA общността – които обединяват усилията си за създаването на стандарт, който да позволи по лесен начин RMI приложения да използват CORBA сървъри и CORBA клиенти да използват RMI-базирани сървърски приложения. Благодарение на взаимни отстъпки на двете общности RMI-IIOP става стандарт, който позволява съвместното използване на разпределени програмни компоненти от света на Java и от света на CORBA, като комбинира много от добрите страни на двете конкурентни технологии – RMI и CORBA.

Благодарение на промените в CORBA стандарта и въвеждането на RMI-IIOP става възможно извикването на CORBA сървъри от Java RMI клиентски приложения по начин, естествен за RMI технологията, без да е необходима никаква промяна в съществуващия CORBA сървър. Става възможно и обратното – CORBA клиенти по лесен начин да извикват RMI-базирани сървъри, по същия начин, по който извикват CORBA сървъри.

Малко след въвеждането му RMI-IIOP става стандартния протокол за комуникация между отделните компоненти на разпределени J2EE приложения.

Технологията RMI

RMI е съкращение от Remote Method Invocation – технология за прозрачна комуникация между Java-базирани приложения. RMI изисква програмистът да опише отдалечените обекти, които ще използва, чрез интерфейси и осигурява възможност по време на изпълнение такива обекти да бъдат търсени, създавани и използвани. Работата с отдалечени RMI обекти не се различава съществено от работата с обикновени Java обекти – разликата е в начина на инстанциране и в наличието на постоянна възможност при извикването на отдалечен метод да възникне изключение поради проблеми с комуникацията. В такъв смисъл комуникацията между двете приложения е прозрачна за програмиста.

RMI има архитектура от тип клиент-сървър. Клиентът използва т. нар. отдалечен интерфейс (remote interface) на обектите от сървъра (java.rmi.Remote), а сървърът реализира обектите, до които клиентът ще има достъп като наследници на класа java.rmi.server.UnicastRemoteObject, които реализират отдалечения интерфейс. По време на компилация на така реализираното клиент-сървър приложение автоматично се генерират специални помощни класове, които се грижат за изпращане и приемане на клиентските заявки за достъп до отдалечените обекти на RMI сървъра. От страна на клиента се генерират прокси-класове (stubs), които маршализират заявките и ги изпращат по протокол JRMP (Java Remote Method Protocol) към сървъра, а от страна на сървъра се генерират сървърски прокси-класове (skeletons), които посрещат идващите от клиента заявки, автоматично активират търсения обект, изпълняват исканата заявка за извикване на метод и връщат обратно към клиента резултата от изпълнението й отново по протокол JRMP. Протоколът GRMP представлява схема за обмяна на съобщения между stubs и skeletons и използва за транспортен слой протокола TCP.

За публикуване и намиране на отдалечени обекти се поддържа специален регистър (RMI registry), в който RMI сървърите регистрират под някакви имена обектите, до които

предлагат отдалечен достъп, а RMI клиентите използват този регистър за да търсят вече регистрирани RMI обекти по името им, под което са регистрирани.

RMI изисква всички обекти (структури от данни), които се предават от клиента към сървъра или обратно да бъдат сериализируеми в смисъла на вградената в Java сериализация. Това осигурява автоматизирано пренасяне на обекти от клиента към сървъра и възможност за репродуцирането им на отдалечената машина. На практика, когато клиентът изпраща като параметър на сървъра обект от непознат за сървъра клас, компилираният Java bytecode на този клас се изпраща на сървъра заедно със сериализираните данни за изпращания обект. Така по време на изпълнение при RMI технологията е възможно да се изпращат не само данни, но и изпълним код, за който сървъра нищо не знае по време на компилация.

Технологията RMI поддържа автоматично разпределено управление на паметта (distributed garbage collection). Това позволява обекти, създадени от клиента по време на неговата работа, да бъдат автоматично унищожавани и да се освобождават ресурсите, използвани от тях, когато настъпи момент, в който те вече не се използват нито от клиента, нито от сървъра.

Протоколът IIOP

IIOP представлява протокол за обмяна на информация между отдалечени приложения, който се използва вътрешно от CORBA (Common Object Request Broker Architecture) и позволява програми написани на различни езици, работещи върху различни хардуерни и софтуерни платформи, да комуникират по TCP/IP-базирани мрежи чрез отдалечено извикване на методи.

IIOP е съкращение от Internet Inter-ORB Protocol и е разработен от Object Management Group (OMG) – организацията, която се грижи за развитието и стандартизацията на технологията CORBA и свързаните с нея стандарти.

IIOP е имплементация на общата спецификация комуникационни протоколи GIOP от CORBA стандарта. Той използва като транспортен слой протокола TCP и осигурява комуникация между ORB-компонентите (Object Request Brokers) на CORBA базирани разпределени приложения. На практика IIOP е основният и най-често използван протокол за пренос на данни в CORBA-базирани разпределени приложения.

IIOP е бинарен протокол и това го прави удобен за приложения, в които производителността е от особено значение.

Нуждата от RMI over IIOP

Първоначално технологиите Java RMI и CORBA са били проектирани да служат сходни цели, но са имали сериозни различия.

Технологията RMI е била предназначена да осигури лесен за програмиста начин за комуникация между две Java приложения и е използвала за връзката между тях протокол JRMP. Интерфейсите на отдалечените обекти са били обикновени Java интерфейси. Било е възможно при извикване на метод без допълнителни усилия да се предават сложни структури от данни, включително и обекти от класове, непознати за сървъра. Благодарение на факта, че и клиента и сървъра са били винаги Java-класове, било е възможно по време на изпълнение да се изпращат не само данните за даден обект, но и неговият компилиран код (Java bytecode).

Целите на CORBA стандарта са били малко по-различни – да осигури взаимодействие между приложения, работещи върху различни платформи и реализирани на различни езици за програмиране. Заради стремежа на CORBA да бъде възможно по-универсална технология, тя е била доста сложна за използване. За описание на интерфейсите на отдалечените обекти е бил използван специален език за дефиниране на интерфейси IDL (Interface Definition Language), който е бил така проектиран, че неговите дефиниции да имат еквивалент във

всички по-разпространени езици за програмиране с универсално приложение (C, C++, Java и др.). При изпращане на сложни структури данни при отдалечено извикване на методи тези структури е било необходимо да бъдат описвани предварително. Не е било възможно да се предаде като параметър структура от данни, която не е известна по време на компилация едновременно и на клиента и на сървъра. CORBA използва протокола IIOP за предаване на заявките и отговорите при извикване на отдалечени методи.

RMI технологията има предимството, че осигурява изключителна леснота на работа – не изисква специално описание на всички обменяни структури от данни, не изисква допълнително описание на интерфейсите на отдалечените обекти във вид на IDL, не изисква програмистът да знае езика IDL и не изисква никакъв допълнителен софтуер, за да бъде използвана.

CORBA има предимството, че позволява взаимодействие между различни езици за програмиране, различни платформи и операционни системи, макар и това да води до известни усложнения като използване на междинен език IDL за описание на интерфейсите и обменяните структури от данни, използване на допълнителен софтуер (имплементация на CORBA стандарта за съответния език за програмиране и платформа) и необходимост от предварително дефиниране на обменяните данни.

Колкото и различни да са световете на RMI и CORBA, тези технологии имат една и съща крайна цел – да осигурят взаимодействие между отделните компоненти на разпределени приложения чрез отдалечено извикване на методи. Идеята да се комбинират двете технологии, като се вземе лесnотата на работа на RMI и възможността за взаимодействие с различни езици за програмиране на CORBA е станала особено привлекателна за големи фирми като Sun и IBM. Благодарение на съвместните усилия на Java общността и CORBA общността в резултат на взаимни отстъпки бил създаден стандартът RMI over IIOP (RMI-IIOP), който осигурил взаимодействие между RMI и CORBA приложения.

Протоколът RMI-IIOP

Протоколът RMI-IIOP е съвместна разработка на Sun и IBM. Той дава възможност RMI клиенти да си комуникират с CORBA сървъри и обратното – CORBA клиенти да си комуникират с RMI сървъри. На практика RMI-IIOP е bridge, който свързва RMI инфраструктурата с IIOP протокола и позволява на RMI приложенията да комуникират по протокол IIOP вместо по стария JRMP. По този начин RMI-IIOP отваря вратите на RMI към взаимодействие на ниско ниво с CORBA приложения.

Между RMI-IIOP и използваният преди това в RMI протокол JRMP има някои разлики, като например липсата на поддръжка на distributed garbage collector при RMI-IIOP.

Имплементация на протокола RMI-IIOP се поддържа от Java още във версия 1.1.6, а от версия 1.3 е стандартна част от Java 2 платформата. Тази имплементация изпълнява пълната функционалност на ORB (Object Request Broker) компонентите от CORBA стандарта.

RMI-IIOP дава възможност на Java програмистите да разработват CORBA приложения без да разбират от особеностите на CORBA, нито от езика IDL. Имайки уменията за работа с RMI, един Java програмист може да използва CORBA, като просто укаже по време на компилация, че иска да използва RMI-IIOP, а не JRMP като преносен протокол за комуникация между отделните компоненти на разпределената система, която разработва.

Възможно е едно и също RMI приложение да използва едновременно и RMI (JRMP) и RMI-IIOP. В такъв случай от страна на сървъра обектите, които ще се използват отдалечено трябва да се експортират двойно (dual export) – веднъж като `UnicastRemoteObject`, който се регистрира за JRMP достъп в RMI регистъра (RMI registry) и веднъж като `PortableRemoteObject`, който се регистрира в CORBA naming услугата CosNaming.

RMI-IIOP в Java 2 платформата

Технологията RMI-IIOP е неделима част от Java 2 платформата. Тя позволява CORBA клиенти да извикват Java RMI сървъри по същия начин, по който извикват CORBA сървъри. В голяма степен е възможно и обратното – Java RMI клиенти да извикват CORBA сървъри по същия начин, по който извикват RMI сървъри. За съжаление не всеки IDL интерфейс има съответствие в RMI-IIOP и затова не винаги е възможно директното използване на CORBA сървъри от RMI клиент.

Компилаторът за RMI приложения **`rmic`** има специална опция **`-iiop`**, която указва дали да се генерират IIOP или JRMP stubs и skeletons. С негова помощ могат да бъдат генерирани и CORBA IDL интерфейси от Java RMI интерфейсите.

RMI (JRMP) или RMI-IIOP

Протоколът RMI-IIOP не е предназначен да замести напълно стария протокол JRMP за комуникация между RMI приложения. Неговата цел е само да разшири възможностите на RMI технологията и да й позволи да си взаимодейства по лесен начин с CORBA. Когато RMI се използва за връзка между две Java приложения, разработчикът има свободата кой от двата протокола да избере – IIOP или JRMP, а ако е необходимо, може да използва едновременно и двата като извърши т. нар. dual export на отдалечените обекти.

Предимствата на RMI-IIOP пред RMI (JRMP) са:

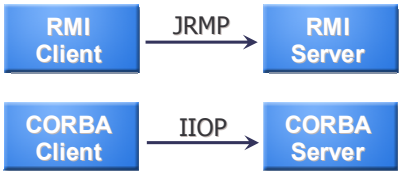
- позволява се взаимодействие с други езици благодарение на CORBA IDL интерфейса, който се генерира по време на компилация и благодарение на протокола IIOP, който CORBA ORB компонентите разбират;
- възможно е при извикване на отдалечен метод автоматично да се подават контекстите за транзакция и за сигурност;
- възможно е заявките да преминават през специализираните IIOP защитни стени (firewalls), което осигурява комуникация и през firewall.

Предимствата на RMI (JRMP) пред RMI-IIOP са:

- поддържа се разпределено автоматично управление на паметта (distributed garbage collection);
- използването на обекти, извлечени от naming услугата, става директно – чрез просто преобразуване на типовете (Java typecasting), без да е необходимо да се извиква специален метод (narrow) за намиране на верния тип;
- константите в отдалечените интерфейси (remote interfaces) могат да бъдат от произволен тип, докато при IIOP могат да бъдат само от примитивни типове (int, long, string, …).

RMI и CORBA – взаимодействие

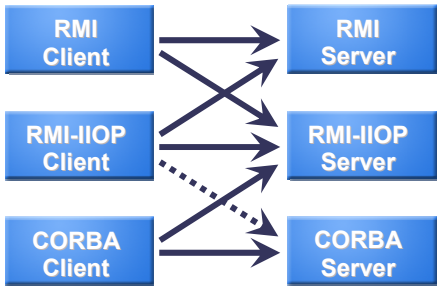
Преди въвеждането на RMI-IIOP възможностите за директно взаимодействие между RMI и CORBA приложения са били доста ограничени:



Client \ Server	RMI (JRMP)	CORBA (IIOP)
RMI (JRMP)	?	X
CORBA (IIOP)	X	?

RMI (JRMP) клиентите могат да си комуникират единствено с RMI (JRMP) сървъри по протокола JRMP. CORBA клиентите могат да си комуникират единствено с CORBA сървъри по протокола IIOP.

След въвеждането на RMI-IIOP възможностите за директно взаимодействие стават много по-богати:



RMI клиентите могат да си комуникират с RMI сървъри вече не само по протокол JRMP, но и по протокол IIOP. В допълнение към това, вече един RMI клиент в повечето случаи може директно да използва CORBA сървъри по протокол IIOP (когато IDL интерфейсът на сървъра е съвместим с RMI-IIOP). CORBA клиентите, освен че могат да извикват директно CORBA сървъри, могат вече да извикват директно и RMI сървъри по протокол IIOP.

Създаването на RMI-IIOP

Резултатът от въвеждането на RMI-IIOP е наистина впечатляващ, но е струвал много усилия на Java и CORBA общностите. Въвеждането на IIOP като общ протокол за предаване на съобщения между RMI и CORBA съвсем не е било достатъчно за да започнат двете технологии съвместна работа. Когато са били разработвани новият стандарт RMI-IIOP и следващата версия на CORBA стандарта, които да позволят директното взаимодействие между RMI и CORBA, Java общността се е стремила да запази програмния модел на RMI възможно най-непроменен. CORBA общността, от своя страна, се е стремила връзката между RMI и CORBA да стане възможна без съществени промени в CORBA стандарта.

Един от проблемите, който е трябвало да бъде решен е как да се предават сложни структури от данни между RMI и CORBA приложения. В CORBA това е ставало чрез предварително дефиниран IDL интерфейс, към който са се преобразували структурите, така че да бъдат разбираеми за всички езици, за които има имплементация на CORBA. В RMI предаването на структури от данни е използвало вътрешните механизми на Java за сериализация на обекти, което е позволявало точното репродуциране на даден обект (структура от данни) на отдалечената машина. За успешното директно взаимодействие между RMI и CORBA е било необходимо двата стандарта да разбират форматите на предаваните между тях данни. Необходимо е било или Java RMI инфраструктурата да промени вътрешния формат на сериализираните Java обекти (или поне начина, по който ги записва в IIOP съобщенията) или

CORBA OBR компонентите е трябвало да започнат да разбират от сериализирани Java обекти. На практика се е случило второто.

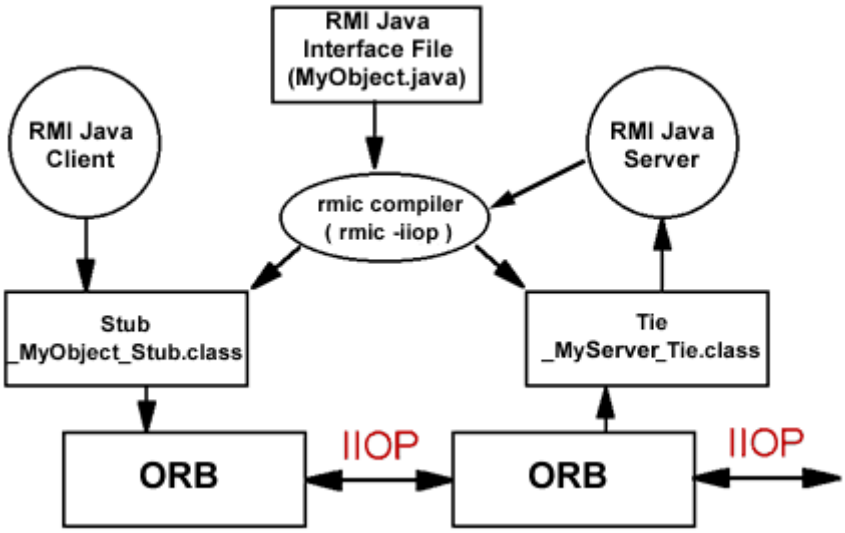
За постигането на съвместимост с Java сериализацията CORBA общността приела две сериозни промени във версия 2.3 на CORBA стандарта. Била добавена възможност за предаване на обекти по стойност (Objects by Value) и е била приета Java-To-IDL Mapping спецификацията.

Objects by Value допълнението към CORBA стандарта позволило при извикване на метод обектите, предавани като параметри да могат да бъдат сериализирани локално и така предавани до отдалечената машина, където да се репродуцира копие от тях. Преди въвеждането на Objects by Values в CORBA стандарта всички примитивни типове били предавани по стойност, а всички съставни типове (структури от данни) били предавани по референция. С новата спецификация станало възможно не само примитивни обекти да се предават по стойност. Така типичният за RMI механизъм за предаване на сериализируеми Java обекти по стойност станал възможен и в CORBA .

Новата Java-To-IDL Mapping спецификация позволила Java RMI интерфейси да бъдат конвертирани към еквивалентни CORBA IDL интерфейси. Така за CORBA приложенията станало възможно да извлекат IDL интерфейса на даден RMI сървър и да чрез него да извикват отдалечено RMI обекти по протокола IIOP.

Разработка на Java приложения с RMI-IIOP

На схемата по-долу е показано как се разработват Java RMI разпределени приложения, които използват протокола RMI-IIOP за обмяна на информация:



Както се вижда, процедурата по създаването на RMI базирано разпределено приложение използващо RMI-IIOP за преносен протокол, е почти същата като процедурата за създаване на RMI приложение, използващо протокол JRMP. Разликите са следните:

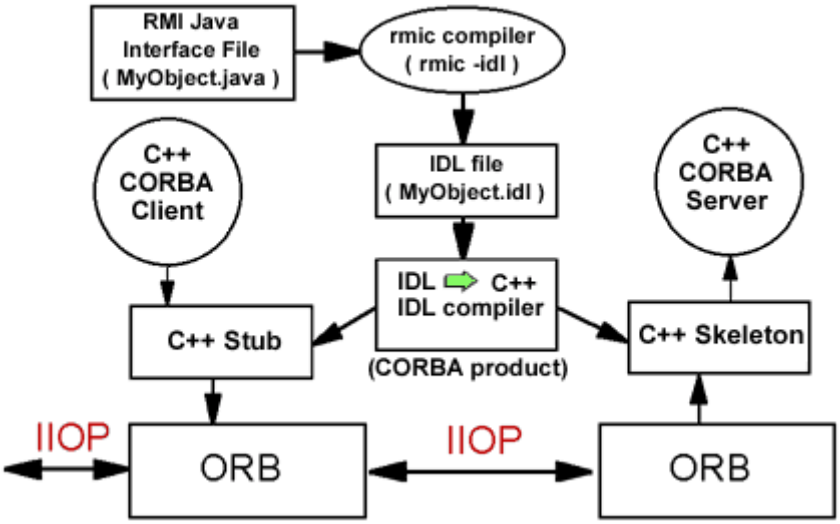
- при генерирането на stub класа и tie класа подава се опцията -iiop на RMI компилатора;
- за реализацията на сървъра се наследява класа PortableRemoteObject вместо UnicastRemoteObject;
- за регистрация и търсене на обекти се използва naming услугата CosNaming вместо RMI registry, достъпна през JDNI `InitialContext` класа;
- за намиране на отдалечения обект в клиента се използва CosNaming услугата вместо RMI registry, достъпна отново през JNDI `InitialContext` класа;

- в клиента се използва `narrow()` метода на `PortableRemoteObject` класа за да се активира намерения обект, вместо да се използва обикновен typecasting.

Първоначално се дефинира отдалеченият интерфейс (remote interface) и се реализират клиента и сървъра, които го използват. След това се компилират клиента и сървъра. Те нямат нужда от stub и tie класовете по време на компилация, но ги използват по време на работа. След това се изпълнява и **`rmic`** компилатора с опция **`-iiop`** за да се получи stub класа на отдалечения интерфейс и tie класа на RMI сървъра. Получените от RMI компилатора stub и tie класове се използват по време на изпълнение от RMI-IIOP инфраструктурата.

Разработка на C++ приложения с RMI-IIOP

На схемата по-долу е показано как се разработват CORBA клиенти и сървъри, които си взаимодействат с Java RMI клиенти и сървъри:



Както се вижда, първоначално Java RMI интерфейсът се преобразува към IDL интерфейс чрез `rmic` компилатора с опция `-idl`. След това разработката на CORBA клиента или сървъра, които използват този IDL интерфейс, става по стандартния за CORBA начин. Чрез някакъв компилатор от IDL към езика за програмиране, който ще се използва, се генерират stub и skeleton класове, които се използват при разработваното CORBA приложението. Накрая готовото вече CORBA приложение се свързва с ORB имплементацията на Java RMI клиента или сървъра и комуникира с него по протокол IIOP.

Перспективи пред IIOP и RMI-IIOP

RMI-IIOP е огромна крачка напред в посока към постигане на съвместимост и лесен начин за директно взаимодействие между Java RMI и CORBA приложения. В тази посока на развитие работата продължава с цел постигане на още по-добра съвместимост на двете технологии, както и съвместимост с други технологии за разработка на разпределени приложения, като например DCOM и .NET Remoting.

RMI-IIOP не решава напълно проблемите със взаимодействието между RMI и CORBA. Все още има някои несъвместимости между тези стандарти, за преодоляването на които може да се работи:

- Java RMI интерфейсите не поддържат всичките възможности на IDL (не всеки IDL има аналог в RMI).
- CORBA не поддържа distributed garbage collection.
- Security моделите на RMI и CORBA се различават.

По отношение на интеграцията между CORBA и .NET Remoting, а оттам и между .NET Remoting и Java RMI вече има постигнати конкретни успехи. Например проектът IIOP.NET свързва .NET Remoting с CORBA и RMI аналогично на RMI-IIOP, а продукти като Borland Janeva имплементират IIOP канал в .NET Remoting и така го отварят към света на CORBA.

По отношения на интеграцията между CORBA и DCOM също има успехи – съществуват bridges, които извършват взаимодействие между двете технологии.

Използвана литература

1. Wollrath A. and Waldo J., The Java Tutorial: RMI – http://java.sun.com/docs/books/tutorial/rmi/
2. TechTarget Network – IIOP Glossary Definition – http://searchnetworking.techtarget.com/gDefinition/0,294236,sid7_gci214019,00.html
3. Java RMI-IIOP Documentation – http://java.sun.com/j2se/1.3/docs/guide/rmi-iiop/
4. Hagge D., RMI-IIOP in the Enterprise – http://www-106.ibm.com/developerworks/java/library/j-rmi-iiop/
5. Andoh A., Nash S., RMI over IIOP – http://www.javaworld.com/javaworld/jw-12-1999/jw-12-iiop.html
6. Programming WebLogic RMI over IIOP – http://edocs.bea.com/wls/docs81/rmi_iiop/index.html
7. Ergul S., Java Primer: RMI Over IIOP – When the Java and CORBA Worlds Collide – http://www.adtmag.com/java/articleold.asp?id=687
8. IIOP.NET – http://iiop-net.sourceforge.net/
9. Borland Janeva – http://www.borland.com/janeva/