

Chapter 17. RMI

Mingyu Lim

**Collaborative Computing Systems Lab,
School of Internet & Multimedia Engineering
Konkuk University, Seoul, Korea**

■ 학습 목표

- RMI란
- RMI 구조
- RMI는 어떻게 동작하는가
- 로컬 객체를 원격 객체로 변경하기
- RMI를 이용한 계산기 애플리케이션
- RMI를 이용한 채팅 애플리케이션
- RMI-IIOP
- RMI-IIOP 예제

RMI란

□ 기존 소켓 프로그래밍

- 서버/클라이언트간 프로토콜을 지키며 통신
- 프로토콜이 복잡해질수록 소켓 프로그래밍 또한 복잡

□ RMI (Remote Method Invocation)

- 원격의 객체 메소드를 로컬컴퓨터의 메소드 호출과 같은 방식
 - ◆ 내부 프로토콜과 상관없이 메소드의 인자와 반환 형식만 맞춰주면 됨
- 유사 방식
 - ◆ CORBA, DCOM, RPC

RMI 구조

□ 로컬 객체 메소드 호출

- Object o = ???
- 같은 JVM의 힙 내에 호출자와 호출대상 존재

□ 원격 객체 메소드 호출

- Object o = ??? 형식 불가
- 호출자와 호출대상이 다른 JVM에 존재

□ RMI

- 클라이언트측 스텝모듈과 서버측 스켈레톤 모듈 통해 원격 객체의 메소드를 로컬 메소드처럼 호출 가능

RMI 구조



□ RMI 스텝/스켈레톤

- 원격 객체 메소드를 로컬 메소드 호출하듯이 처리
- 클라이언트에서 스텝의 메소드 호출
- 스텝은 원격의 서버측 스켈레톤에 접속하여 메소드명, 인자 전달
- 서버는 스켈레톤 통해 요청 메소드정보를 알아내어 처리
- 반환값은 다시 스켈레톤을 통해 클라이언트 스텝으로 전달

□ 스텝/스켈레톤은 J2SDK 내의 rmic컴파일러로 생성

RMI는 어떻게 동작하는가

□ RMI 통신 순서

- 1) 서버가 원격 객체를 RMI Registry에 등록
- 2) 클라이언트가 RMI Registry에서 원격 객체 Lookup
- 3) 클라이언트가 원격 객체 이용하여 원격 메소드 호출
- 서버에서 등록 객체가 스텝

로컬객체를 원격객체로 변환

- 원격객체를 이용한 Hello 애플리케이션 작성 순서
 - 1) 원격 인터페이스 작성
 - 2) 원격 인터페이스 실제 구현하는 원격 클래스 작성
해당클래스는 `java.rmi.server.UnicastRemoteObject` 상속
 - 3) 원격 객체를 `rmiregistry`에 등록하기 위한 애플리케이션 작성
 - 4) 원격 객체를 이용하는 클라이언트 작성
 - 5) 스텝과 스켈레톤 생성 (`rmic` 이용)
 - 6) 원격 객체를 `rmiregistry`에 등록
 - 7) 클라이언트 애플리케이션 실행

로컬객체를 원격객체로 변환

□ 1) 원격 인터페이스 작성

- RMI 원격 인터페이스 상속 (java.rmi.Remote)
- 인터페이스 내 메소드는 반드시 RemoteException을 throws
- 메소드에 전달하는 인자와 반환값은 기본형이거나 java.io.Serializable 인터페이스 구현한 것

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMHello extends Remote
{
    public String hello(String name) throws RemoteException;
}
```


로컬객체를 원격객체로 변환

□2) 원격인터페이스를 구현하는 원격 객체 작성

- Java.rmi.server.UnicastRemoteObject 상속
- 원격 인터페이스에 있는 메소드 실제 구현
- Java.rmi.RemoteException을 throws하는 기본생성자
- 원격 인터페이스에 있는 메소드 구현하는 경우
java.rmi.RemoteException을 throws하게 한다

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RMHelloImpl extends UnicastRemoteObject
implements RMHello
{
    public RMHelloImpl() throws RemoteException{}
    public String hello(String name) throws RemoteException
    {
        return "Hello "+name+" !";
    }
}
```

로컬객체를 원격객체로 변환

□3) 원격객체를 등록하기 위한 애플리케이션 작성

```
import java.rmi.Naming;  
import java.rmi.RMISecurityManager;
```

```
public class RMISHelloBind  
{
```

```
    public static void main(String[] args)  
    {
```

```
        RMISecurityManager sm = new RMISecurityManager();  
        System.setSecurityManager(sm);
```

```
        try{
```

```
            RMISHelloImpl obj = new RMISHelloImpl();  
            Naming.rebind("//127.0.0.1:2000/hello", obj);
```

```
            System.out.println("Hello객체가 Registry에 등록되었습니다.");
```

```
        }catch(Exception e){
```

```
            System.out.println("등록이 안되었습니다." + e.toString());
```

```
        }
```

```
    }
```

```
}
```

신뢰할수없는
서버로부터 스
텝적재시 자신
을 보호하기 위
해 사용

Rmiregistry
에 Hello이름
으로 등록

로컬객체를 원격객체로 변환

□4) 원격 객체를 실제로 이용하는 클라이언트 작성

```
import java.rmi.Naming;
```

```
public class RMHelloClient
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        try{
```

```
            Object obj = Naming.lookup("//127.0.0.1:2000/hello");
```

```
            RMHello hi = (RMHello)obj;
```

```
            String msg = hi.hello("김성박");
```

```
            System.out.println(msg);
```

```
        }catch(Exception e){
```

```
            System.out.println(e);
```

```
        }
```

```
    }
```

```
}
```

Hello원격객체 lookup

RMHello인터페이스로 형변환

로컬객체 메소드처럼 사용

로컬객체를 원격객체로 변환

□5) 컴파일과 스텝, 스켈레톤 생성

- 소스 컴파일
 - ◆ `Javac *.java`
- 스텝/스켈레톤 생성
 - ◆ `rmic RMHelloImpl` (class file있는 디렉토리에서)
- 클래스파일 분리
 - ◆ Client 디렉토리
 - `RMHello.class`, `RMHelloImpl.class`, `RMHelloClient.class`,
`RMHelloImpl_Stub.class`
 - ◆ Server 디렉토리
 - `RMHello.class`, `RMHelloImpl.class`, `RMHelloBind.class`,
`RMHelloImpl_Skel.class`(or `RMHelloImpl_Stub.class`)

로컬객체를 원격객체로 변환

□6) rmiregistry 실행과 rmiregistry에 원격객체 등록 (Server 디렉토리)

– rmiregistry 실행

◆ rmiregistry 2000 (2000포트로 실행)

– Policy 파일 작성

```
grant {  
    //Allow everything for now  
    permission java.security.AllPermission;  
};
```

– 원격객체 등록

◆ Java -Djava.security.manager -Djava.security.policy=policy
RMHelloBind

```
D:\Data\Program\NIOsource\17-4RMHello\server>java -Djava.security.manager -Djav  
a.security.policy=policy RMHelloBind  
Hello객체가 Registry에 등록되었습니다.
```

로컬객체를 원격객체로 변환

□7) 원격객체 이용하는 클라이언트 실행 (클라이언트 디렉토리)

– java RMHelloClient

```
D:\Data\Program\NIOSource\17-4RMHello\client>dir
D 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 2813-191B

D:\Data\Program\NIOSource\17-4RMHello\client 디렉터리

2009-08-26 오후 02:25 <DIR> .
2009-08-26 오후 02:25 <DIR> ..
2009-08-26 오후 01:48          230 RMHello.class
2009-08-26 오후 02:11       1,010 RMHelloClient.class
2009-08-26 오후 01:53          722 RMHelloImpl.class
2009-08-26 오후 02:24       1,799 RMHelloImpl_Stub.class
                4개 파일              3,761 바이트
                2개 디렉터리 129,073,082,368 바이트 남음

D:\Data\Program\NIOSource\17-4RMHello\client>java RMHelloClient
Hello 김성박 !

D:\Data\Program\NIOSource\17-4RMHello\client>_
```

RMI를 이용한 계산기 애플리케이션

□ 4직연산 가능한 애플리케이션 작성 순서

- 1)계산기 원격 인터페이스 작성
- 2)계산기 원격 인터페이스를 구현하는 원격클래스 작성
java.rmi.server.UnicastRemoteObject 상속
- 3)원격객체를 rmiregistry에 등록하는 애플리케이션 작성
- 4)원격객체를 이용하는 클라이언트 작성
- 5)스텝과 스켈레톤 생성 (rmic이용)
- 6)원격객체를 rmiregistry에 등록
- 7)클라이언트 애플리케이션 실행

RMI를 이용한 계산기 애플리케이션

□1)계산기 원격 인터페이스 작성

```
import java.rmi.*;

public interface Calculator extends Remote
{
    public int sum(int i, int j) throws RemoteException;
    public int multiply(int i, int j) throws RemoteException;
    public int divide(int i, int j) throws RemoteException,
    ArithmeticException;
    public int subtract(int i, int j) throws RemoteException;
}
```


RMI를 이용한 계산기 애플리케이션

□2)계산기 원격 인터페이스를 구현하는 원격클래스 작성

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculatorImpl extends UnicastRemoteObject implements
Calculator
{
    public CalculatorImpl() throws RemoteException{}
    ...
    public int divide(int i, int j) throws RemoteException,
ArithmeticException
    {
        if(j == 0)
            throw new ArithmeticException("0으로 나눌수 없습니다.");
        return i / j;
    }
    public int subtract(int i, int j) throws RemoteException
    {
        return i - j;
    }
}
```

RMI를 이용한 계산기 애플리케이션

□3)원격객체를 rmiregistry에 등록하는 애플리케이션 작성

```
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class CalculatorBind
{
    public static void main(String[] args)
    {
        RMISecurityManager sm = new RMISecurityManager();
        System.setSecurityManager(sm);
        try{
            CalculatorImpl obj = new CalculatorImpl();
            Naming.rebind("//127.0.0.1:2000/calculator", obj);
            System.out.println("Calculator객체가 Registry에 등록되었습니다.");
        }catch(Exception e){
            System.out.println("등록이 안되었습니다." + e.toString());
        }
    }
}
```

RMI를 이용한 계산기 애플리케이션

□ 4)원격객체를 사용하는 클라이언트 작성

– Calculator 원격객체 lookup

```
Object obj = Naming.lookup("//127.0.0.1:2000/calculator");  
Calculator cal = (Calculator)obj;
```

– 로컬메소드 호출 방식으로 사용

```
int value = 0;  
if(args[1].equals("+"))  
{  
    value = cal.sum(op1, op2);  
}else if(args[1].equals("-"))  
{  
    value = cal.subtract(op1, op2);  
}else if(args[1].equals("*"))  
{  
    value = cal.multiply(op1, op2);  
}else if(args[1].equals("/"))  
{  
    value = cal.divide(op1, op2);  
}
```

RMI를 이용한 계산기 애플리케이션

□5)스텝/스켈레톤 생성

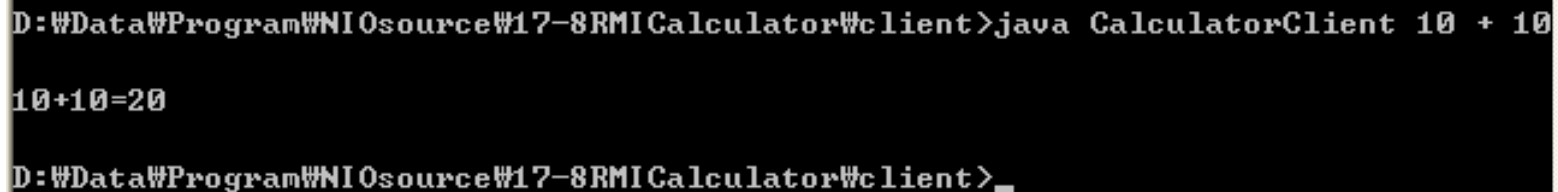
- rmic CalculatorImpl

□6)원격객체를 rmiregistry에 등록 (server 디렉토리)

- Policy파일 작성
- rmiregistry 2000 (포트2000 에 registry실행)
- Java -Djava.security.manager -Djava.security.policy=policy CalculatorBind (등록)

□7)클라이언트 실행 (client 디렉토리)

- Java CalculatorClient 정수 연산자 정수



```
D:\Data\Program\NIOsource\17-8RMI\Calculator\client>java CalculatorClient 10 + 10
10+10=20
D:\Data\Program\NIOsource\17-8RMI\Calculator\client>_
```

RMI를 이용한 채팅 애플리케이션

□로그인

- 클라이언트가 서버의 connect() 호출
- 서버는 connect()가 호출되면 모든 접속 사용자에게 로그인사용자 정보를 전달
 - ◆ 클라이언트의 print() 원격 메소드 호출
 - ◆ 서버도 클라이언트측의 원격메소드 호출 (RMI콜백)

□대화

- 클라이언트는 별도의 스레드로 사용자 입력 처리
- 사용자 입력 문자열은 서버의 broadcast()호출하여 다른 클라이언트들에게 전달
- 서버는 broadcast()호출되면, 접속 클라이언트들의 print() 원격 메소드 호출하여 문자열내용 전달

□로그아웃

- 클라이언트는 서버의 disconnect() 원격 메소드 호출
- 서버는 접속클라이언트의 print()호출하여 로그아웃사용자 정보 전달

RMI-IIOP

- RMI와 코바의 장점을 결합한 기업용 전송기술
- RMI와 코바의 차이
 - 코바: IIOP 이용, RMI: JRMP이용
 - 코바: 언어독립적, RMI: 자바에서만 사용
 - 코바: 원격객체의 위치찾기 위해 CosNaming이용, RMI: JNDI이용
 - 코바: 객체직렬화기술 사용안함, RMI: 객체직렬화기술 사용
- RMI-IIOP의 RMI 개발과의 차이
 - Rmiregistry가 아닌 tnameserv실행
 - Rmic 실행시 iiop옵션 지정하여 iiop형식의 스텝과 타이 생성
 - 원격객체는 PortableRemoteObject 상속
 - 원격객체 등록,검색을 위해 Connect 객체 사용
 - Policy 파일 필요없음

RMI-IIOP 예제

□ RMI-IIOP를 이용한 Hello

- 1)원격인터페이스 작성
- 2)원격인터페이스를 구현하는 원격클래스 작성
javax.rmi.PortableRemoteObject 상속
- 3)원격객체를 네이밍서버(tnameserv)에 등록하는
애플리케이션 작성
- 4)원격객체를 이용하는 클라이언트 작성
- 5)스텝과 타이 생성(rmic에 iiop옵션 지정)
- 6)원격객체를 네이밍서버(tnameserv)에 등록
- 7)클라이언트 애플리케이션 실행

RMI-IIOP 예제

□1) 원격인터페이스 작성

- Java.rmi.Remote 인터페이스 확장
- 메소드는 모두 java.rmi.RemoteException throws
- 인자와 반환값은 모두 직렬화가능 형식

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface HelloIIOP extends Remote {  
    public String hello() throws RemoteException;  
}
```


RMI-IIOP 예제

□2) 원격인터페이스를 구현하는 원격클래스 작성

- Javax.rmi.PortableRemoteObject 상속
- 원격인터페이스의 메소드 구현
 - ◆ Java.rmi.RemoteException throws
- Java.rmi.RemoteException throws하는 기본 생성자 필요

```
import java.rmi.RemoteException;
import javax.rmi.PortableRemoteObject;

public class HelloIIOPImpl extends PortableRemoteObject
implements HelloIIOP {
    public HelloIIOPImpl() throws RemoteException {}
    public String hello() throws RemoteException {
        return "Java World";
    }
}
```

RMI-IIOP 예제

□3) 원격객체를 네이밍서버(tnameserv)에 등록하는 애플리케이션 작성

– HelloIIOPImpl객체 등록

```
HelloIIOPImpl obj = new HelloIIOPImpl();  
java.util.Properties p = new java.util.Properties();  
p.put(Context.INITIAL_CONTEXT_FACTORY,  
       "com.sun.jndi.cosnaming.CNCtxFactory");  
p.put(Context.PROVIDER_URL, "iiop://localhost");  
Context ctx = new InitialContext(p);  
ctx.rebind("helloiiop", obj);
```

- ◆ Properties객체에 정보 지정하여 InitialContext생성자의 인자로 지정
- ◆ InitialContext는 내부적으로 com.sun.jndi.cosnaming.CNCtxFactory클래스 이용하여 네이밍서버에 접속
- ◆ 접속시 iiop프로토콜 사용하여 localhost에 접속

RMI-IIOP 예제

□4) 원격객체를 이용하는 클라이언트 작성

– 객체 lookup (Context객체 이용)

```
java.util.Properties p = new java.util.Properties();  
p.put(Context.INITIAL_CONTEXT_FACTORY,  
       "com.sun.jndi.cosnaming.CNCtxFactory");  
p.put(Context.PROVIDER_URL, "iiop://localhost");  
Context ctx = new InitialContext(p);  
Object obj = ctx.lookup("helloiiop");
```

– HelloIIOP인터페이스 형변환 (PortableRemoteObject의 narrow()이용)

```
HelloIIOP h =  
(HelloIIOP)PortableRemoteObject.narrow( obj, HelloIIOP.class);
```

– 원격객체 호출

```
System.out.println(h.hello());
```

RMI-IIOP 예제

□5) 스텝과 타이 생성

– Rmic -iiop HelloIIOPImpl

□6) 원격객체를 네이밍서버(tnameserv)에 등록

– tnameserv 실행

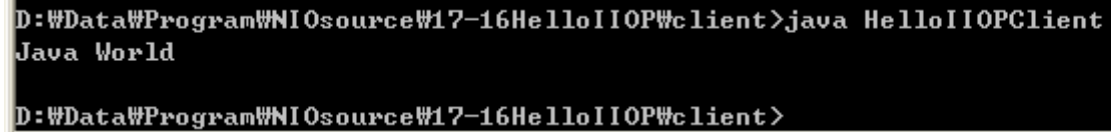
```
D:\Data\Program\NIOSource\17-16HelloIIOPI\server>tnameserv
초기 명령 컨텍스트:
IOR:0000000000000002b49444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f
6e746578744578743a312e3000000000000100000000000009e00010200000000103230332e3235
322e3138322e313033000384000000000045afabcb0000000020000f42400000000100000000000
000200000008526f6f74504f41000000000d544e616d6553657276696365000000000000080000
0001000000011400000000000020000000100000020000000000001000100000002050100010001
002000010109000000010001010000000026000000020002
TransientNameServer: 초기 객체 참조를 위한 포트 설정: 900 900
준비되었습니다.
```

– 원격객체 등록

```
D:\Data\Program\NIOSource\17-16HelloIIOPI\server>java HelloIIOPIBind
등록되었습니다.
```

RMI-IIOP 예제

□7) 클라이언트 애플리케이션 실행



```
D:\Data\Program\NIOsource\17-16HelloIIOP\client>java HelloIIOPClient
Java World
D:\Data\Program\NIOsource\17-16HelloIIOP\client>
```