JNDI

- Abbiamo già visto come i registri RMI espletino un servizio di *Naming* attraverso cui vengono associati nomi simbolici a referenze a server remoti.

- Esistono comunque altri servizi di naming:

  - COS (Common Object Service) Naming: il servizio di naming per applicazioni CORBA.
  - DNS (Domain Name System): il servizio di naming per Internet. Associa indirizzi simbolici come ad esempio

    `profs.sci.univr.it`

    ad indirizzi IP della forma

    157.27.242.115.

  - LDAP (Lightweight Directory Access Protocol): un servizio standard di naming per le reti.

- NIS (Network Information System) e NIS+: È un servizio di naming di rete, sviluppato dalla Sun Microsystems, per consentire agli utenti di accedere un qualunque file in rete attraverso un ID ed una password.

- JNDI (Java Naming and Directory Interface) è un'*interfaccia* che consente di accedere in modo uniforme servizi di naming differenti come quelli elencati sopra.

- L'API di JNDI è indipendente dal servizio di naming supportato. Le uniche differenze notate da un client JNDI sono:

  - Che dovrà specificare quale servizio di naming vuole implementare;

  - La possibilità che una certa operazione non sia supportata dal servizio di naming supportato.

- I concetti fondamentali su cui si basa JNDI sono i seguenti:

  - *Name*: il nome che viene dato all'oggetto registrato

- *Binding*: l'associazione nome-oggetto
- *Reference*: un puntatore ad un oggetto
- *Context*: insieme di associazioni nome-oggetto. Ad esempio in un filesystem tipo Unix un file directory "/usr" è un context. Un oggetto context può essere legato ad un altro oggetto context chiamato subcontext. Ad esempio un file directory "/usr/bin" è un subcontext.
- *Naming System*: è un insieme di context dello stesso tipo.
- *Namespace*: è l'insieme di nomi in un naming system.

• Similmente alla classe `Naming` vista precedentemente, l'interfaccia `javax.naming.Context` fornisce i metodi *bind, unbind, rebind, lookup, list.*

• Oltre a questi metodi, vi sono, tra l'altro, un metodo *rename* per rinominare un oggetto, e un metodo *listBindings*, simile alla list, che oltre agli oggetti ritorna anche le rispettive classi di appartenenza.

JNDI Providers

- JNDI i servizi di naming non vengono acceduti direttamente, ma attraverso plugin chiamati *provider*.
- I servizi di naming acceduti tramite provider forniscono alcuni vantaggi, quali:
  - *persistenza*, ovvero i binding sono salvati e ripristinati in caso di reinizializzazione del sistema;
  - *distribuzione* su una rete di calcolatori.
- È importante capire che il provider JNDI non è "il servizio di naming" ma solo un'interfaccia ad un qualche servizio di naming. Cioè il servizio di naming è dato da un server esterno. Ad esempio, quando si accede un registro RMI attraverso JNDI è comunque necessario lanciare il registro RMI.
- Un provider JNDI viene settato fissando le seguenti due proprietà di sistema:

– `java.naming.factory.initial`

– `java.naming.provider.url`.

- Come d'abitudine queste due proprietà possono essere istanziate da programma oppure da linea di comando, atraverso l'opzione -D, al momento del lancio dell'applicazione.

RMI registry provider

- Un provider JNDI per un registro RMI media tra l'API di JNDI ed il registro RMI.

- Ad esempio, volendo settare un provider JNDI per un registro RMI sulla macchina `profs.sci.univr.it` alla porta 4567, bisognerà assegnare alle due proprietà sopra menzionate i seguenti valori, rispettivamente:

  – `com.sun.jndi.rmi.registry.RegistryContextFactory`

  – `rmi://profs.sci.univr.it:4567` (il valore di default è `//localhost:1099`).

- Ma perchè complicarsi la vita con JNDI invece di usare direttamente un registro RMI?

- Il vantaggio di JNDI è che consente di supportare diverse forme di naming. Così se in seguito si vuole passare ad un servizio di naming diverso, come ad esempio LDAP il codice vero e proprio non cambia, a parte il settaggio di JNDI.

COS Naming provider

- JNDI fornisce un'interfaccia anche per il servizio di naming COS per servi scritti in CORBA.

- Tale servizio di naming dovrà essere utilizzato nel caso in cui il nostro codice Java RMI contenga server remoti `PortableRemoteObject` esportati attraverso RMI/IIOP (li vedremo nella prossima lezione).

- In tal caso alla proprietà `java.naming.factory.initial` va assegnato il valore `com.sun.jndi.cosnaming.CNCtxFactory` mentre la proprietà `java.naming.provider.url` prende valori del tipo: `iiop://arena.sci.univr.it:4040`

- Similmente è possibile configurare il provider JNDI con altri servizi di naming quali LDAP.

Esempio di lookup su registro RMI

Una tipica operazione di lookup da parte di un client su un registro RMI sappiamo avere la seguente forma:

```
import java.rmi.Naming;
........
IntRemota obj =
  (IntRemota)Naming.lookup(``//profs.sci.univr.it:4567/ServerRemoto'')
```

La medesima operazione usando JNDI diventa:

```
import javax.naming.*;
...........
Properties prop new Properties();
prop.put(Context.INITIAL_CONTEXT_FACTORY,
         ``com.sun.jndi.rmi.registry.RegistryContextFactory'');
prop.put(Context.PROVIDER_URL,
         ``rmi://profs.sci.univr.it:4567'');
Context cxt = new InitialContext(prop);
IntRemota obj =
  (IntRemota)cxt.lookup(``profs.sci.univr.it:4567/ServerRemoto'');
```

- Nota che in questo esempio le due proprietà di sistema sono state assegnate da programma. Un'altra possibilità era quella di definirle da linea di comando con l'opzione -D.

- `Context` è un'interfaccia che specifica un Naming System cioè un insieme di associazioni nome-oggetto. Questa interfaccia fornisce i metodi per aggiungere, cancellare, ridenominare, cercare, ... oggetti.

- `InitialContext` è un'implementazione dell'interfaccia `Context` e rappresenta il contesto di partenza per eseguire operazioni di naming. Il valore dell'environment iniziale viene preso come argomento del costruttore.

- `static String INITIAL_CONTEXT_FACTORY` è una costante a cui viene assegnata la proprietà di sistema che individua il servizio supportato.

- `PROVIDER_URL` è una costante a cui viene assegnato l'URL del servizio di naming usato.