Java Naming and Directory Interface (JNDI)

Alma Mater Studiorum - Università di Bologna
CdS Laurea Magistrale in Ingegneria Informatica
I Ciclo - A.A. 2017/2018
Corso di Sistemi Distribuiti M (8 cfu)

04 - Appendice su Sistemi di Nomi in Ambiente Enterprise

Docente: Paolo Bellavista
paolo.bellavista@unibo.it

http://lia.deis.unibo.it/Courses/sd1718-info/
http://lia.deis.unibo.it/Staff/PaoloBellavista/

Introduzione: Naming, Discovery, Directory

Ruolo cruciale del servizio di nomi in ogni sistema distribuito, ancora di più per ogni sistema di livello enterprise

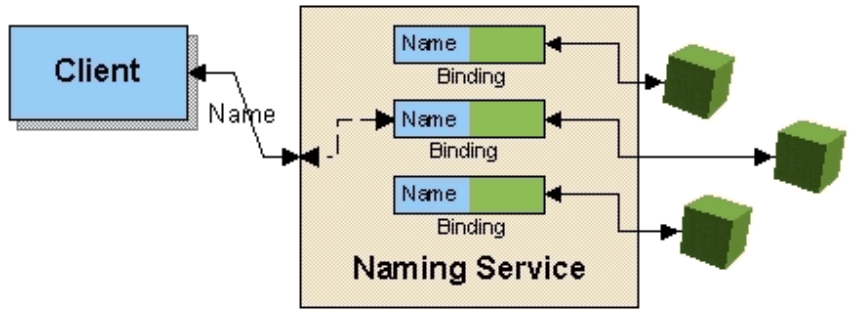
Vi ricordate, vero ☺, che cosa si intende per:

- ***sistema di naming?***
  - quali funzionalità? Esempi?
- ***protocollo di discovery?***
  - quali funzionalità? In quali scenari? Esempi?
- ***sistema di directory?***
  - quali funzionalità? Esempi? Quali differenze rispetto ai precedenti?

Introduzione: Naming

Naming service mantiene un ***insieme di binding fra nomi e oggetti*** (o riferimenti a oggetti – ad es. DNS)

Java Naming & Directory Interface (JNDI) come interfaccia che supporta ***funzionalità comuni*** ai vari differenti sistemi di nomi



Introduzione: Naming

JNDI come ***workable abstraction*** che permette di lavorare con servizi di nomi diversi nonostante loro differenze

Ad esempio, ***differenze in naming convention:***

- in DNS, nomi come composizione di elementi separati da punti e letti da destra verso sinistra – www.unibo.it
- in LDAP, situazione leggermente più complessa. Nomi composti da elementi separati da virgole, letti da destra a sinistra, ma specificati come coppie nome/valore - "cn=Paolo Bellavista, o=UniBO, c=IT"

Per esempio a livello di schemi/formati diversi di naming, JNDI risolve problema dell'eterogeneità attraverso classe **`Name`**, le sue sottoclassi e classi helper. **`Name`** come rappresentazione di un nome composto da una sequenza non ordinata di sequenze di subname

Introduzione: Discovery

Directory
- soluzioni di nomi globali
- servizi completi e complessi
- costo elevato delle operazioni

Discovery
- soluzioni di ***nomi locali***
- ***servizi essenziali e funzioni limitate***
- ***costo limitato*** adatto a ***variazioni rapide***

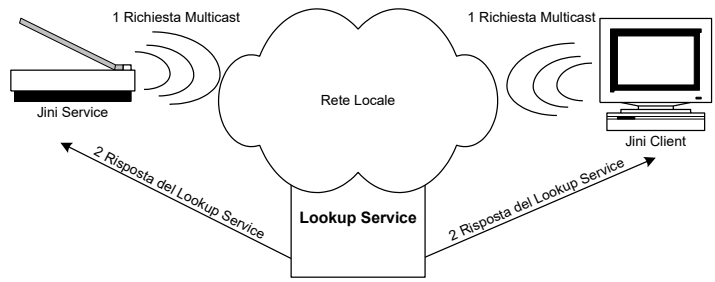
Directory per informazioni globali, come descrizione dispositivi, preferenze profili utente, firme digitali e PKI, sorgenti di informazioni, ecc.

Discovery per informazioni locali, come descrizione risorse locali gestori presenti, ecc.

Introduzione: Discovery

Directory garantisce proprietà di replicazione, sicurezza, gestione multiserver, ... Supporto per memorizzare le informazioni organizzate prevedendo ***molti accessi in lettura e poche variazioni***

Invece ***protocolli di Discovery***
Computazione distribuita e cooperativa ***in ambito di località***
Una unità deve ritrovarne altre, in modo ***veloce ed economico***
Si prevedono azioni come broadcast locale e solleciti periodici

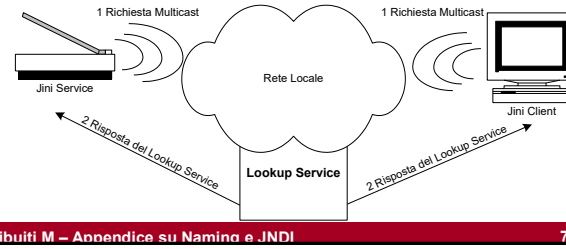


Introduzione: Discovery

JINI (ma anche UPnP, SSDP, SLP, …)
protocollo Java per discovery: si vuole rispondere alle esigenze di chi arriva in un contesto e ***vuole operare senza conoscenze predefinite***

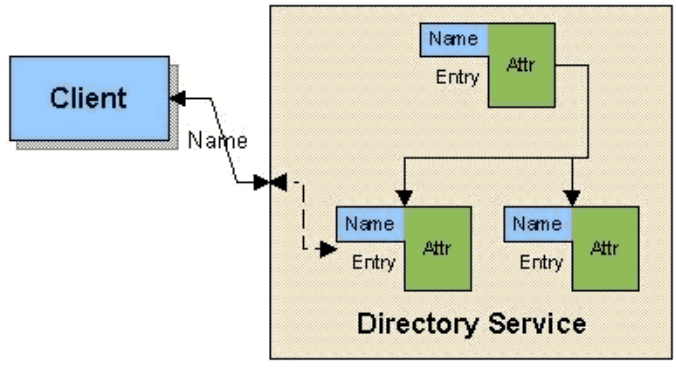
Protocolli di lookup
Il server può passare (tramite lookup) al cliente:
- anche codice (che si può eseguire localmente)
- riferimento al server (da interrogare in remoto - RMI)

Start up con multicast in ambiente locale
Discovery server verifica presenza risorse a intervalli opportuni



Introduzione: Directory

Directory service come strumento per ***gestire storage e distribuzione di info condivise***: da indirizzi email a numeri di telefono degli impiegati di un'azienda, a indirizzi IP e proprietà di stampanti di un dipartimento, da info di configurazione a un insieme di application server



Introduzione: Directory

Directory come gestore di un ***insieme (directory) di record (entry)***, ***conformi a un determinato schema*** (directory schema), riferite a persone, luoghi, servizi, oggetti concreti o concetti astratti

Entry hanno attributi associati: attributi hanno nome/identificatore e uno o più valori. Ad esempio, per un individuo:

Name: John Doe
Address: 123 Somewhere Street
Email: john@xyz.com
Email: jdoe@abcd.com

Directory come semplici ***database specializzati, spesso con funzionalità di search&filter***

Naming: name-to-object mapping

Directory: informazioni su oggetti e strumenti per ricerca avanzata

Lightweight Directory Access Protocol (LDAP) come comune esempio di directory con funzionalità di naming+directory (implem. OpenLDAP)

Introduzione: Java Naming & Directory Interface (1)

Esistono molti servizi di naming:

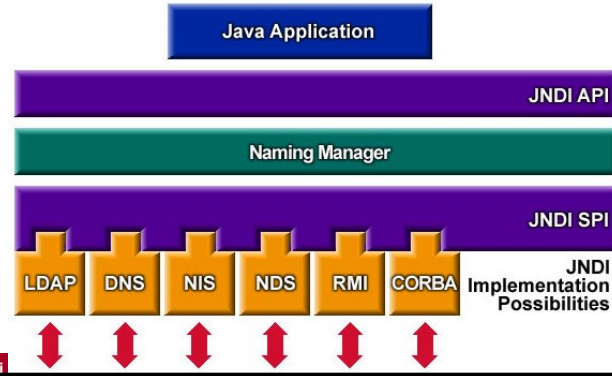
- ***Registry RMI*** – associazione fra nomi simbolici e riferimenti a server remoti
- ***Common Object Service (COS) naming*** – naming standard per applicazioni CORBA
- ***Domain Name System (DNS)*** – associazione fra indirizzi simbolici e indirizzi IP fisici
- ***Lightweight Directory Access Protocol (LDAP)*** – naming standard Internet per direttori risorse (persone, risorse di rete, …)
- ***Network Information System (NIS)*** – naming di rete proprietario di Sun Microsystems per accedere a file system di rete (ID e password di accesso)

Introduzione: Java Naming & Directory Interface (2)

Java Naming and Directory Interface (JNDI) come interfaccia ***per accedere in modo uniforme*** a servizi di naming differenti ***(già esistenti!)***

API JNDI indipendente dal servizio di naming supportato

- Att.ne: possibilità che una certa operazione ***non sia supportata dal servizio di naming dinamicam. collegato a JNDI***



Introduzione: Java Naming & Directory Interface (3)

Concetti fondamentali:

Name – nome dato all'oggetto registrato

Binding – associazione nome-oggetto

Reference – puntatore a un oggetto

Context – insieme di associazioni nome-oggetto. Ad es. in un filesystem Unix-like, una directory può essere un context, che contiene ***sub-context***

Naming system – un insieme di context tutti dello stesso tipo

Namespace – insieme di nomi in un naming system

JNDI Provider

In JNDI i ***servizi di naming vengono acceduti attraverso plugin chiamati (name service) provider***

Provider JNDI NON è servizio di naming ma un'interfaccia di connessione verso uno specifico servizio di naming ***esterno*** (in terminologia JNDI, è cliente JNDI verso vero servizio di nomi esterno)

Provider JNDI si occupa di ***supporto alla persistenza*** (binding salvati e nel caso ripristinati dopo un fault) ***e distribuzione su rete***

Essenzialmente, provider JNDI scelto configurando le proprietà

- `java.naming.factory.initial`
- `java.naming.provider.url`

JNDI Provider

Ad esempio, in caso di registry RMI, occorre configurare un JNDI provider per registry RMI ***(mediatore fra JNDI e registry)***:

proprietà initial `com.sun.jndi.rmi.registry.RegistryContextFactory`

proprietà url `rmi://lia.deis.unibo.it:5599`

Perché usare JNDI invece di usare direttamente un registry RMI?

Vantaggio di JNDI: consente di supportare diverse forme di naming, senza modificare codice cliente, ***a parte setting JNDI***

Esempio di Lookup su Registro RMI

Vedremo in molti esempi che tipica operazione di lookup ha l'aspetto:

```
import java.rmi.Naming;
…
IntRemota obj = (IntRemota) Naming.lookup
        ("//lia.deis.unibo.it:5599/ServerRemoto");
```

Invece utilizzando JNDI:

```
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://lia.deis.unibo.it:5599");
Context cxt = new InitialContext();
IntRemota cxt.lookup("lia.deis.unibo.it:5599/ServerRemoto");
```

Context è interfaccia che specifica Naming System, con metodi per aggiungere, cancellare, cercare, ridenominare, … oggetti

InitialContext è un'implementazione di Context e rappresenta contesto di partenza per operazioni di naming

JNDI: Context

Ora scendiamo in qualche dettaglio più preciso…

Context e *InitialContext*

Classe ***Context*** svolge un ruolo centrale in JNDI. ***Context*** rappresenta ***insieme di binding*** all'interno di un servizio di nomi e che ***condividono stessa convenzione di naming***. Oggetto Context per fare ***binding/unbinding di nomi a oggetti, per fare renaming e per elencare binding correnti***

Alcuni servizi di nomi supportano ***subcontext*** (context dentro un altro context, come cartella in direttorio)

Tutte le operazioni di naming in JNDI sono svolte in relazione a un context. Si parte da una classe ***InitialContext***, istanziata con proprietà che definiscono tipo di servizio di nomi in uso (eventualm. ID e password)

JNDI: Context

Interfaccia Context (vi ricordate RMI Naming?):

- `void bind(String stringName, Object object)` – nome non deve essere associato già ad alcun oggetto
- `void rebind(String stringName, Object object)`
- `Object lookup(String stringName)`
- `void unbind(String stringName)`
- `void rename(String stringOldName, String stringNewName)`
- `NamingEnumeration listBindings(String stringName)`: restituisce enumeration con nomi del context specificato, insieme a oggetti associati e loro classi

In generale, anche per altre API: ***anche versione con Name object invece di String*** (classe Name per manipolare nomi in modo generico, senza conoscere lo specifico naming service in uso)

Esempio (1)

Come connettersi a naming service, elencare tutti binding o uno specifico, usando filesystem come servizio di nomi

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.Binding;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import java.util.Hashtable;

public class Main {
  public static void main(String [] rgstring) {
    try { // Crea contesto iniziale. Environment specifica
          // quale JNDI provider utilizzare e a quale URL
     Hashtable hashtableEnvironment = new Hashtable();
     hashtableEnvironment.put(Context.INITIAL_CONTEXT_
       FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
```

Esempio (2)

```
…
hashtableEnvironment.put(Context.PROVIDER_URL, rgstring[0]);
Context context = new InitialContext(hashtableEnvironment);

if (rgstring.length == 1) {
       NamingEnumeration namingenumeration =
          context.listBindings("");
        while (namingenumeration.hasMore()) {
           Binding binding =(Binding)namingenumeration.next();
           System.out.println(binding.getName() + " " +
             binding.getObject(); } }
else { for (int i = 1; i < rgstring.length; i++) {
         Object object = context.lookup(rgstring[i]);
         System.out.println(rgstring[i] + " " + object); } }
      context.close(); }
    catch (NamingException namingexception) {
      namingexception.printStackTrace(); }
  } }
```


JNDI: DirContext

DirContext, sottoclasse di Context, estende le funzionalità standard di naming con altre relative a attributi e ricerche su entry di directory

`void bind(String stringName, Object object, Attributes attributes)`

Associa un nome a un oggetto e memorizza gli attributi specificati nella entry corrispondente (preserva attributi esistenti)

`void rebind(String stringName, Object object, Attributes attributes)`

Elimina l'eventuale binding precedente

`DirContext createSubcontext(String stringName, Attributes attributes)`

Crea un sottocontesto, eventualmente con attributi

JNDI: DirContext

```
Attributes getAttributes(String stringName)

Attributes getAttributes(String stringName, String []
       rgstringAttributeNames)
```

Restituisce gli attributi associati con l'entry specificata; Attributes è una classe che modella una collection di Attribute (prima versione – tutti gli attributi; seconda versione – gli attributi specificati nell'array fornito)

```
void modifyAttributes(String stringName, int nOperation,
       Attributes attributes)

void modifyAttributes(String stringName, ModificationItem []
       rgmodificationitem)
```

Modifica gli attributi associati all'entry specificata. Operazioni consentite ADD_ATTRIBUTE, REPLACE_ATTRIBUTE e REMOVE_ATTRIBUTE. Prima versione – stessa operazione su diversi attributi, seconda versione – serie di operazioni su uno o più attributi

DirContext: funzionalità avanzate

Ricerca sul nome dell'attributo

```
NamingEnumeration search(String stringName, Attributes
attributesToMatch)

NamingEnumeration search(String stringName, Attributes
attributesToMatch, String [] rgstringAttributesToReturn)
```

Ricerca in un unico context le entry che contengono un dato insieme di attributi. Prima versione – intero insieme di attributi restituito; seconda versione – restituzione di un insieme scelto di attributi

Ricerca attraverso ***filtro RFC 2254 (filtri per ricerche LDAP espressi come stringhe)***

```
NamingEnumeration search(Name stringName, String
stringRFC2254Filter, SearchControls searchcontrols)

NamingEnumeration search(Name stringName, String
stringRFC2254Filter, Object [] stringRFC2254FilterArgs,
SearchControls searchcontrols)
```

DirContext: funzionalità avanzate

Oggetto SearchControls controlla gli aspetti cruciali della ricerca:

```
SearchControls(int nSearchScope, long nEntryLimit, int
nTimeLimit, String [] rgstringAttributesToReturn, boolean
boolReturnObject, boolean boolDereferenceLinks)
```

nSearchScope: configura lo scope di ricerca al solo livello corrente (OBJECT_SCOPE), anche al livello immediatamente sottostante (ONELEVEL_SCOPE), o all'intero sotto-albero (SUBTREE_SCOPE)

nEntryLimit: ***massimo numero di entry*** che possono essere restituite

nTimeLimit: ***massimo numero di ms*** per la durata della ricerca

rgstringAttributesToReturn: ***quali attributi restituire*** associati alle entry risultato

boolReturnObject: se gli oggetti collegati alle entry devono essere restituiti

boolDereferenceLinks: ***riferimenti devono essere dereferenziati?*** Un link può riferire altre entry di un directory ed estendersi su diversi sistemi di nomi

JNDI e Tecnologie Java

JNDI gioca un ruolo centrale in molte tecnologie Java-related, ad esempio JDBC, JMS e EJB

- JDBC (prima tecnologia per DB relazionali). Uso di JNDI a partire da JDBC 2.0 Optional Package, in congiunzione con interfaccia ***DataSource***. Un oggetto DataSource specifica nome della sorgente, driver da caricare e utilizzare, sua locazione; connessione alla sorgente dati in modo trasparente rispetto a dettagli implementativi DB. ***La specifica JDBC richiede l'uso di JNDI per memorizzare oggetti DataSource***
- La specifica JMS descrive ***oggetti administered*** (che contengono info configurazione JMS e sono usati da clienti JMS per trovare specifici queue/topic). ***La specifica impone di trovare JMS administered object tramite JNDI***
- In EJB2.x, oggetto EJBHome deve essere pubblicato via JNDI. ***In EJB3.0 uso trasparente e pervasivo di JNDI***

JNDI e Tecnologie Java

Perché JNDI così centrale nell'utilizzo di molte tecnologie Java?

JNDI promuove il concetto e l'utilizzo di una sorgente di informazione (naming system) ***idealmente centralizzata – requisito chiave di gestione in molti ambienti enterprise***

- Maggiore semplicità di amministrazione
- Maggiore semplicità per clienti di trovare l'informazione desiderata (sicuro riferimento a unico servizio)

Un Esempio un po' più Complesso e Completo

Primo passo: ottenere initial context

1. Occorre ***scegliere un naming service provider*** (ad es. OpenLDAP o un'altra implementazione di LDAP). Bisogna aggiungere il ***nome del provider all'insieme di proprietà di ambiente*** (in un oggetto Hashtable):

```
Hashtable hashtableEnvironment = new Hashtable();

hashtableEnvironment.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
```

2.Aggiungere ***ogni info addizionale necessaria al naming provider*** (ad es. per LDAP, URL che identifica il servizio, context radice, nome e password per connessione):

```
hashtableEnvironment.put(Context.PROVIDER_URL,
       "ldap://localhost:389/dc=etcee,dc=com");
```

…

Un Esempio un po' più Complesso e Completo

…

```
hashtableEnvironment.put(Context.SECURITY_PRINCIPAL, "name");
hashtableEnvironment.put(Context.SECURITY_CREDENTIALS,
       "password");
```

3. Ottenere ***initial context***. Se si intendono eseguire solo operazioni di naming, oggetto ***Context***; altrimenti oggetto ***DirContext***. ***Non tutti naming provider li supportano entrambi***

```
Context context = new InitialContext(hashtableEnvironment);
```

oppure

```
DirContext dircontext = new InitialDirContext
       (hashtableEnvironment);
```

Un Esempio un po' più Complesso e Completo

Usare servizio di nomi per ***lavorare con oggetti***

Spesso la possibilità di memorizzare oggetti Java nel servizio di nomi è molto utile: ***memorizzazione di oggetti in memoria stabile e loro condivisione*** fra applicazioni diverse o fra differenti esecuzioni della stessa applicazione

Semplicissimo!

```
context.bind("name", object);
```

- Vi ricordate vero la semantica RMI?

Analogia con RMI, ma ***semantica definita meno chiaramente***: ad esempio, consentito che operazione bind() memorizzi o uno ***snapshot dell'oggetto*** o un ***riferimento a un oggetto "live"***

Un Esempio un po' più Complesso e Completo

Operazione duale di lookup:

```
Object object = context.lookup("name")
```

Il punto è che la specifica JNDI lascia a JNDI service provider la ***definizione della semantica dell'operazione di memorizzazione di un oggetto***. "Encouraged (but not required) to support object storage in one of the following formats:"

- Dati serializzati
- Riferimento
- Attributi in un directory context

Un Esempio un po' più Complesso e Completo

Come serialized data

Memorizzazione della rappresentazione serializzata di un oggetto (requisito che classe implementi interfaccia Serializable). Molto semplice!

Ad esempio per mettere una LinkedList in una entry JNDI:

```
// creazione linked list
  LinkedList linkedlist = new LinkedList();
// bind
  context.bind("cn=foo", linkedlist);
// lookup
  linkedlist = (LinkedList)context.lookup("cn=foo");
```

Un Esempio un po' più Complesso e Completo

Come riferimento

A volte ***non è appropriato (o addirittura impossibile) usare la serializzazione***. Ad esempio, oggetto che fornisce un servizio distribuito – di solito si è interessati a trovare info necessarie per trovare oggetto e come comunicare con esso. Altro esempio: connessione a risorsa esterna a JVM, come database o file

=> ***oggetto Reference corrispondente o classe che implementi interfaccia Referenceable*** (oggetto genera un'istanza di Reference quando richiesto). Reference contiene info per ricreare il riferimento, ad esempio un oggetto File che punti al file corretto

Un Esempio un po' più Complesso e Completo

Come attributi

Se name service provider con funzionalità directory, anche memorizzazione di un oggetto come ***collezione di attributi***

- Oggetto deve includere codice per ***scrivere il suo stato interno come un oggetto Attributes***
- Si deve anche fornire ***factory di oggetti per ricostruzione*** dell'oggetto a partire dagli attributi
- Approccio utile quando l'oggetto deve ***poter essere acceduto anche da applicazioni non-Java***

Su Configurazione di Environment

JNDI è una interfaccia generica e generale

Per accedere a uno specifico naming/directory service, occorre specificare quale service provider utilizzare, quale server, …

Configurazione attraverso proprietà di environment:

- ***Standard***
- ***Service-specific***
- ***Feature-specific***
- ***Provider-specific***

Proprietà di ambiente standard

Indipendenti da service provider, anche se non presenti su tutti

Prefisso "java.naming.“. Ad esempio:

- java.naming.batchsize

  Dimensione preferita nella restituzione dei risultati

  Costante: Context.BATCHSIZE; *Default*: fornito dal provider

Su Configurazione di Environment

… oppure

- java.naming.factory.url.pkgs

Lista di prefissi di package, separati da virgole, per le factory

Ad es. `vendorZ.jndi.ldap.ldapURLContextFactory`

Queste classi devono implementare interfacce ObjectFactory o DirObjectFactory

Prefisso di package "com.sun.jndi.url" è sempre in fondo alla lista

Questa proprietà è utilizzata quando un URL viene passato ai metodi di InitialContext

Costante: Context.URL_PKG_PREFIXES; *Default*: lista vuota

- java.naming.provider.url

URL per la configurazione del service provider specificato dalla proprietà "java.naming.factory.initial“

Costante: Context.PROVIDER_URL; *Default*: fornito dal provider

Su Configurazione di Environment

Proprietà di ambiente service-specific

Comuni per tutti naming service provider che implementano un determinato ***servizio o protocollo standard***, ad es. LDAP

Hanno prefisso "java.naming.*service*.“, ad es. "java.naming.ldap.”

Proprietà di ambiente feature-specific

Comuni per tutti naming service provider che implementano ***una specifica feature***, ad es. SASL per autenticazione

Hanno prefisso "java.naming.*feature*.“, ad es. "java.naming.security.sasl.”

Proprietà di ambiente provider-specific

Specifiche per un determinato naming service provider, ad es. servizio Sun LDAP ha una proprietà per abilitare tracing

Ovviamente con prefisso unico, ad es. "com.sun.jndi.ldap.trace.ber"

Su Configurazione di Environment

Come specificare proprietà di ambiente

Attraverso ***parametro environment*** passato al costruttore di InitialContext (ma anche tramite ***file application resource***, ***proprietà di sistema*** e ***parametri di applet***)

File Application Resource

Un file application resource ha ***nome jndi.properties e contiene una lista di coppie attributo/valore*** secondo il formato java.util.Properties

JNDI considera automaticamente i file application resource nei classpath delle applicazioni e in JAVA_HOME/lib/jndi.properties

Ad es. questo programma scorre i nomi in un contesto senza specificare proprietà di ambiente nel costruttore:

```
InitialContext ctx = new InitialContext();
NamingEnumeration enum = ctx.list("");
```

Modalità semplice e che permette di configurare JNDI per tutte le applicazioni che usano lo stesso context, specificando un unico file JAVA_HOME/lib/jndi.properties

Su Configurazione di Environment

Proprietà di sistema

Una proprietà di sistema è una ***coppia attributo/valore*** che Java runtime definisce/usa per descrivere utenti, ambiente di sistema e JVM

Insieme di default di proprietà di sistema. Altre proprietà possono essere aggiunte tramite ***invocazione con opzione -D*** a linea di comando. Ad es:

```
# java -Dmyenviron=abc Main
```

La classe java.lang.System ha metodi statici per leggere e modificare proprietà di sistema. ***JNDI prende le seguenti proprietà dalle proprietà di sistema:***

```
java.naming.factory.initial        java.naming.factory.object
java.naming.factory.state          java.naming.factory.control
java.naming.factory.url.pkgs       java.naming.provider.url
java.naming.dns.url
```

```
Hashtable env=new Hashtable(); env.put(Context.APPLET, this);
// Passa l'ambiente al costruttore di initial context
Context ctx = new InitialContext(env);
```

Su Configurazione di Environment

Quindi, riassumendo, le proprietà di ambiente possono essere specificate tramite ***file application resource, il parametro environment, le proprietà di sistema, e parametri di applet***

Nel caso di utilizzo contemporaneo di una pluralità di questi meccanismi:

- prima ***unione di parametro environment e tutti file application resource***
- poi regola aggiuntiva che alcune ***proprietà standard*** possono essere prese da ***parametri di applet e proprietà di sistema***

Nel caso di ***proprietà presenti in più sorgenti***, generalm. valori delle proprietà sono concatenati in una lista separata da virgole; per alcune proprietà viene preso solo primo valore assegnato