Lezione 15– Gestione sicurezza

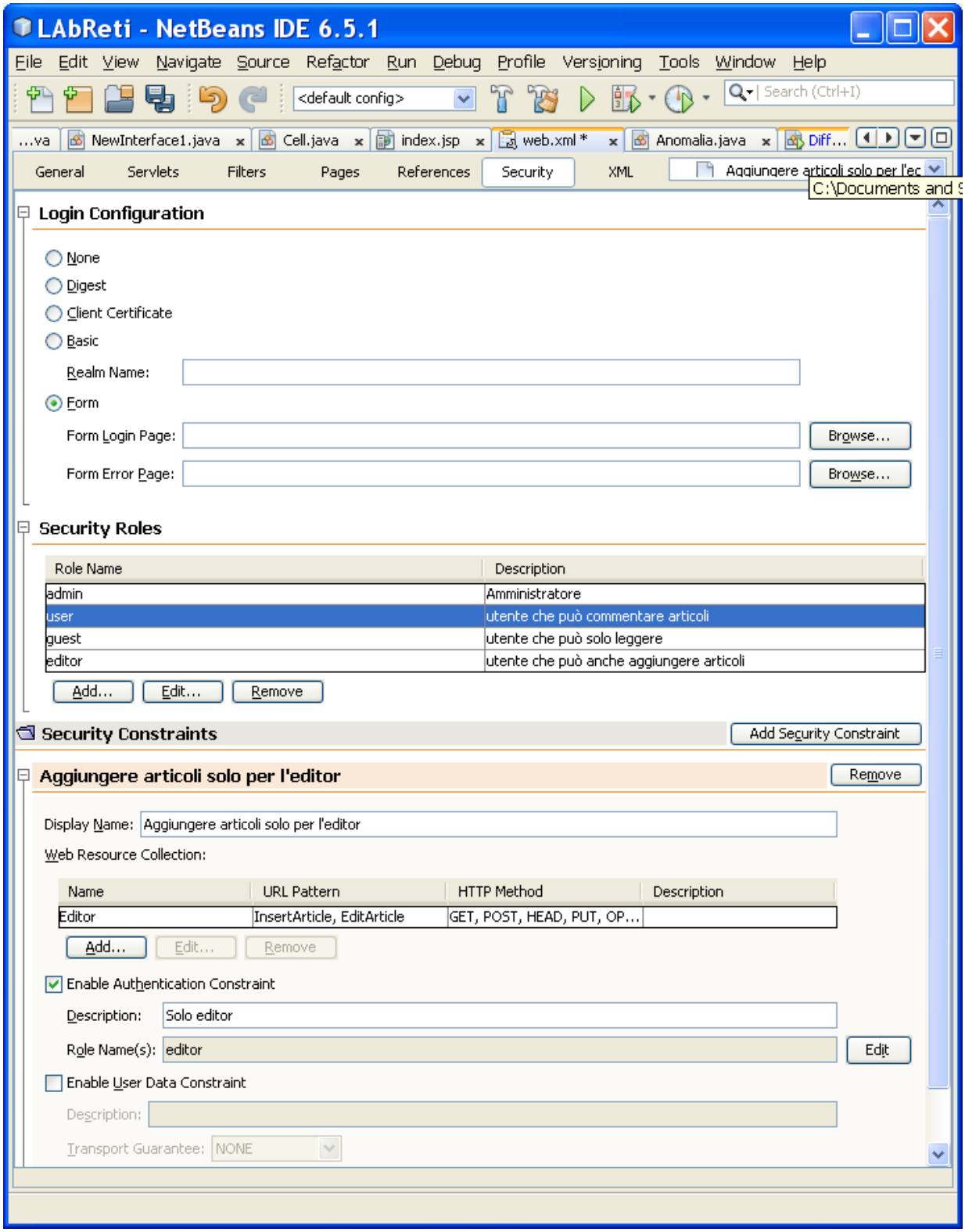
Gestione accessi tramite motore delle servlet

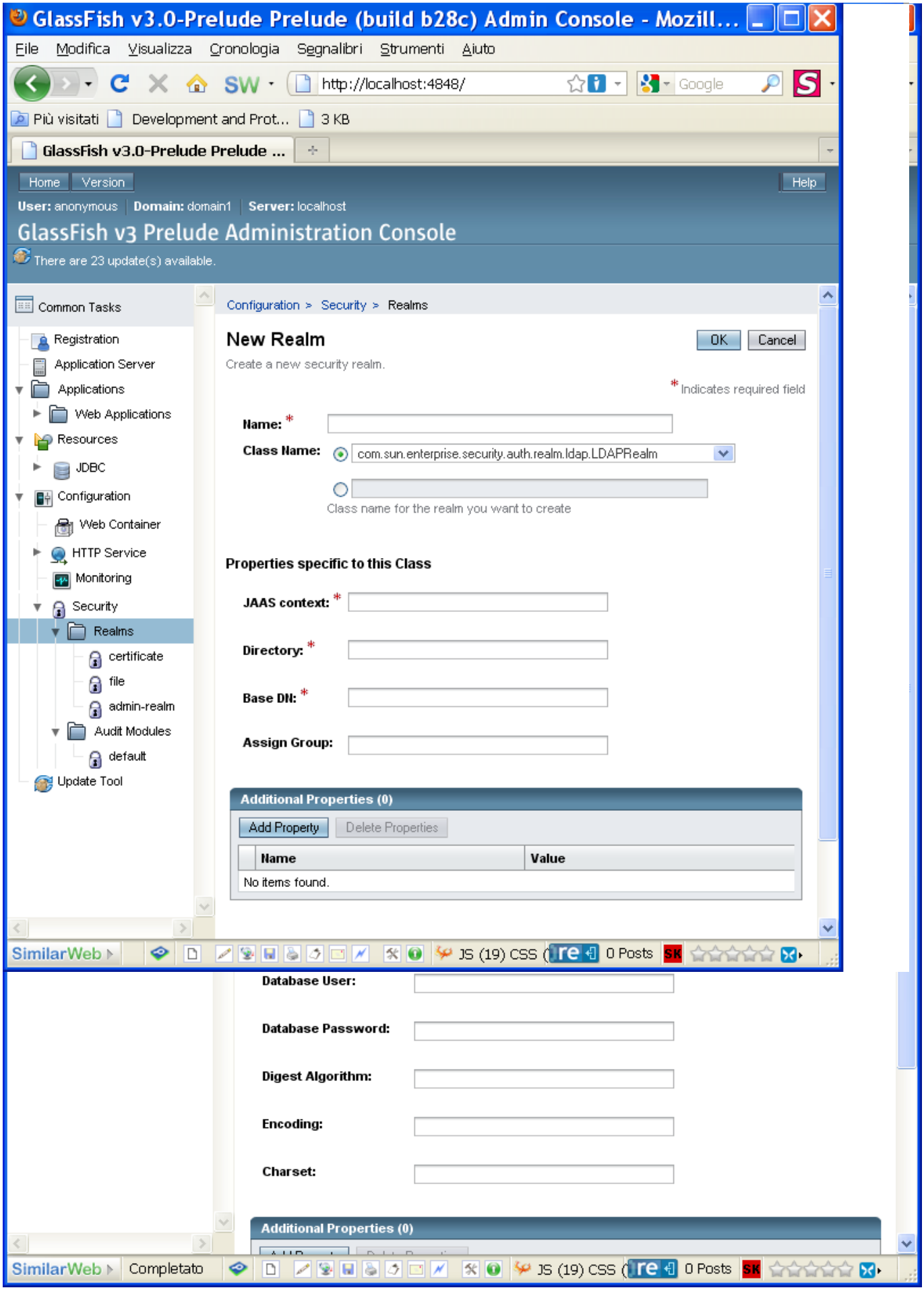
Gli accessi all'applicazione Web possono essere gestiti in automatico dal motore delle servlet. Sono necessarie due configurazioni: una nell'applicazione (file web.xml) e una nel server (dipendente dal server).

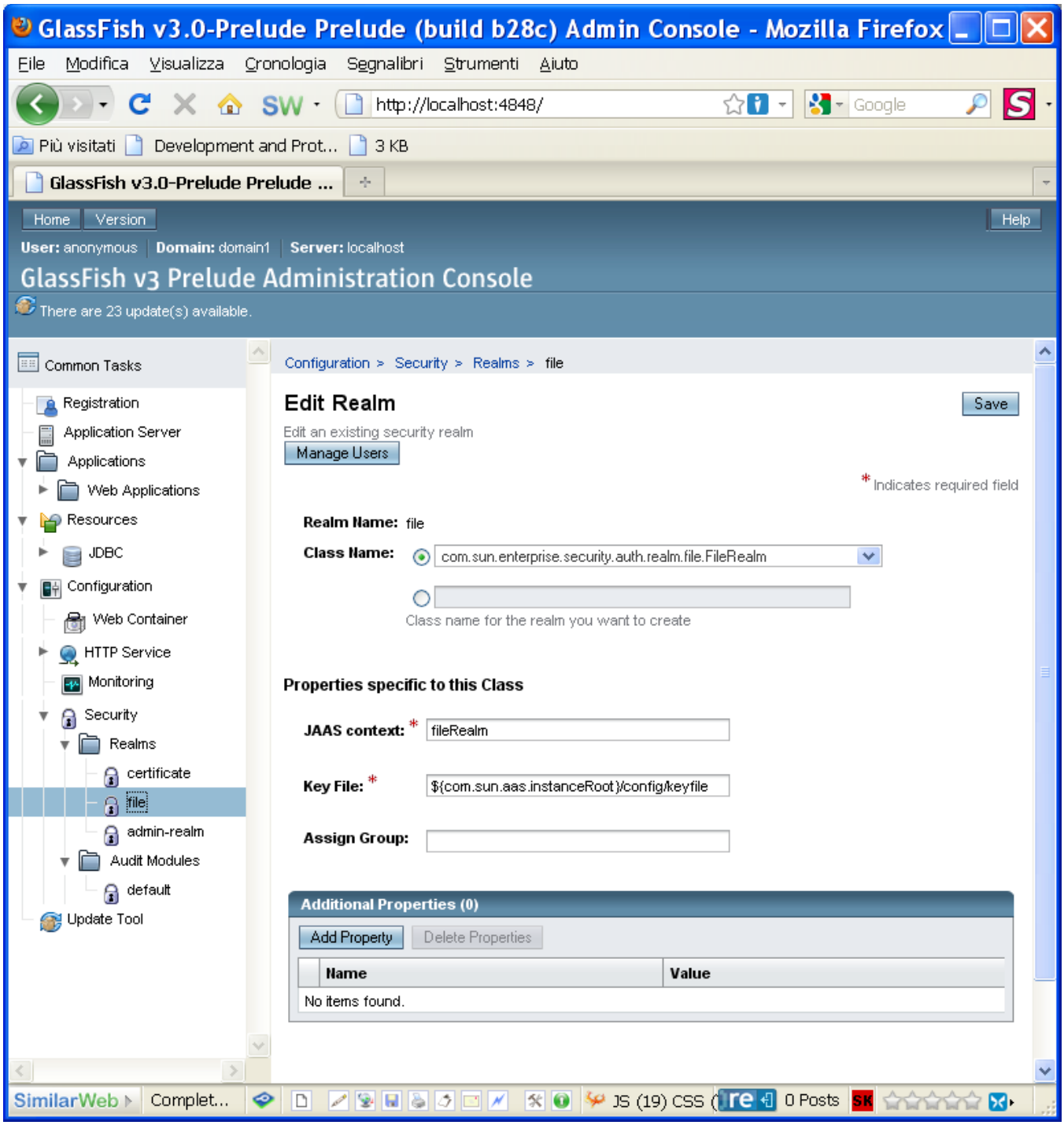
Nell'applicazione si decide quali:

1. come fare il login
2. quali sono i ruoli
3. che risorse possono essere viste dai vari ruoli

Nel server si decide in quale modo recuperare i dati degli utenti (login, password e ruolo di ogni utente). Questi dati possono essere recuperati in vari modi: file di testo, DB o LDAP.


LAbReti - NetBeans IDE 6.5.1
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Search (Ctrl+I)
NewInterface1.java
Cell.java
index.jsp
web.xml *
Anomalia.java
Diff...
General Servlets Filters Pages References Security XML
Aggiungere articoli solo per l'ec
C:\Documents and S
Login Configuration
None
Digest
Client Certificate
Basic
Realm Name:
Form
Form Login Page:
Browse...
Form Error Page:
Browse...
Security Roles
Role Name
Description
admin
Amministratore
user
utente che può commentare articoli
guest
utente che può solo leggere
editor
utente che può anche aggiungere articoli
Add...
Edit...
Remove
Security Constraints
Add Security Constraint
Aggiungere articoli solo per l'editor
Remove
Display Name: Aggiungere articoli solo per l'editor
Web Resource Collection:
Name
URL Pattern
HTTP Method
Description
Editor
InsertArticle, EditArticle
GET, POST, HEAD, PUT, OP...
Add...
Edit...
Remove
Enable Authentication Constraint
Description: Solo editor
Role Name(s): editor
Edit
Enable User Data Constraint
Description:
Transport Guarantee: NONE


GlassFish v3.0-Prelude Prelude (build b28c) Admin Console - Mozill...
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
http://localhost:4848/
Google
Più visitati
Development and Prot...
3 KB
GlassFish v3.0-Prelude Prelude ...
Home
Version
Help
User: anonymous
Domain: domain1
Server: localhost
GlassFish v3 Prelude Administration Console
There are 23 update(s) available.
Common Tasks
Registration
Application Server
Applications
Web Applications
Resources
JDBC
Configuration
Web Container
HTTP Service
Monitoring
Security
Realms
certificate
file
admin-realm
Audit Modules
default
Update Tool
Configuration > Security > Realms
New Realm
OK
Cancel
Create a new security realm.
* Indicates required field
Name: *
Class Name:
com.sun.enterprise.security.auth.realm.ldap.LDAPRealm
Class name for the realm you want to create
Properties specific to this Class
JAAS context: *
Directory: *
Base DN: *
Assign Group:
Additional Properties (0)
Add Property
Delete Properties
Name
Value
No items found.
SimilarWeb
JS (19) CSS (
0 Posts
SK
Database User:
Database Password:
Digest Algorithm:
Encoding:
Charset:
Additional Properties (0)
SimilarWeb
Completato
JS (19) CSS (
0 Posts
SK



Cenni JNDI

Come un database server, anche un directory server memorizza e gestisce delle informazioni ma, a differenza da un database relazionale general purpose, in un directory server:

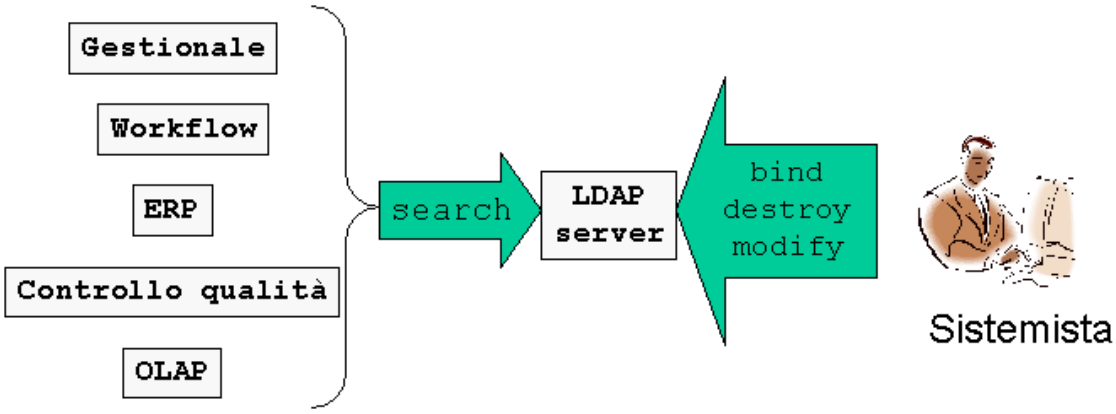
- le ricerche (letture) sono molto maggiori rispetto alle scritture;
- non è adatto a gestire informazioni che variano molto di frequente;
- tipicamente non vengono supportate le transazioni ;
- è previsto un linguaggiod'interrogazione più semplice dell'SQL.

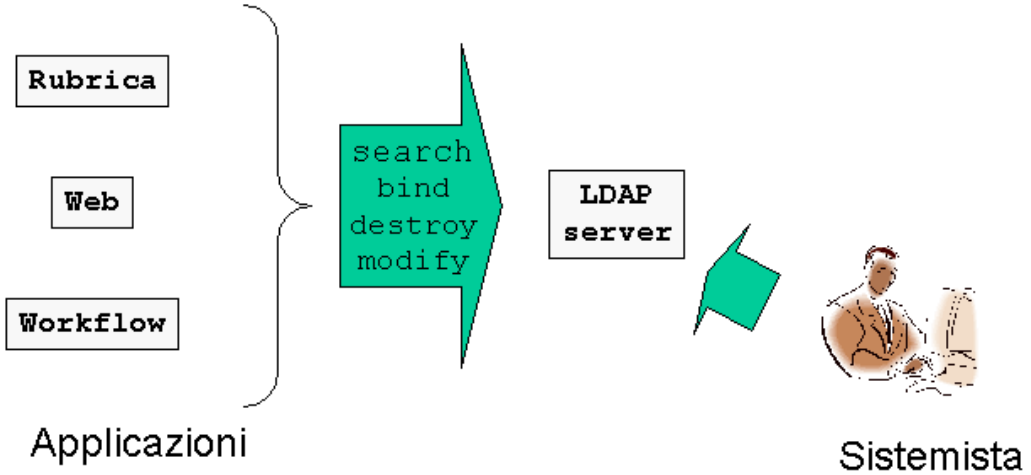
La ricerca delle informazioni è privilegiata per due ordini di motivi:

1. le ricerche sono veloci;
2. le applicazioni che possono beneficiare della ricerca (lettura) in un server LDAP sono molte, mentre le applicazioni che devono modificare i dati in un server LDAP sono poche;

Gerarchia dei nomi

Un nome LDAP può essere pensato come un path di un file in una cartella del sistema operativo. Quindi un nome è specificato da una gerarchia che è adatto in tutti quei casi in cui l'applicazione modella una gerarchia.





Per accedere a un server LDAP, JAVA fornisce l'API JNDI che permette di accedere a un generico servizio di Naming e Directory. Questo significa che alcune operazioni per accedere a LDAP non sono

così semplici e dirette come potrebbero esserlo usando un API specifica per LDAP.
Le operazioni fatte dall'API JNDI per accedere a un LDAP server sono le seguenti:

```
java.net.Socket soc = new
java.net.Socket("ldap.dsi.unive.it",389);

java.io.OutputStream out = soc.getOutputStream();
java.io.InputStream in = soc.getInputStream();

compose(request); // OK semplice

out.write(request);
in.read(response);

parse(response); // Difficile
```

Ecco il codice della classe Ldap che fornisce i principali servizi per accedere a un server LDAP usando JNDI.

```
import javax.naming.directory.*;
import javax.naming.*;

public class Ldap
{
  public static String driver = "com.sun.jndi.ldap.LdapCtxFactory";

  public final String AUTH_SIMPLE="simple";

  DirContext ctx = null;

  public void connect(String ldapUrl) throws NamingException
  {
    //creazione di una hashtable vuota
    java.util.Hashtable env = new java.util.Hashtable();

    //Specifica del driver da usare
    env.put(Context.INITIAL_CONTEXT_FACTORY, driver);

    //Specifica del LDAP server da contattare
    env.put(Context.PROVIDER_URL, ldapUrl);

    //caricamento driver e collegamento al server
    ctx = new directory.InitialDirContext(env);

  }

  public NamingEnumeration search(String base, String filtro,
SearchControls vincoli)
    throws NamingException
  {
   try{
     return ctx.search(base,filtro,vincoli);
   }
   catch(NameNotFoundException nnfe)
   {
     return null;
```

```
   }
  }
...
}
```

Esempio d'uso:

```
try
{
Ldap ldap = new Ldap();

ldap.connect("ldap:/ldap.dsi.unive.it:389");
}
catch(NamingException ne)
{ne.printStackTrace();}

String base = "dc=unive,dc=it";
String filtro = "cn=*";

SearchControls vincoli= new SearchControls();
vincoli.setSearchScope(SearchControls.SUBTREE_S
COPE);

...= ldap.search(base,filtro,vincoli);
```

Sintassi dei Search Filter

La ricerca dei dati è filtrata attraverso una stringa con sintassi articolata, prima di vedere tutti gli elementi della sintassi ne vediamo un esempio e il suo significato.

```
(& (sn=Roncato) (mail=*) ( ! (cn=Alessandro Roncato)) )
```

Il significato è il seguente: un'entry che abbia l'attributo mail presente, il surname sia Roncato e il commonName non sia Alessandro Roncato.
Come si può vedere, sono già standardizzati dei concetti che aiutono a esprimere ricerche che riguardano gli utenti.
La query è composta da test su degli attributi, questi test sugli attributi compongono i **termini** del linguaggio che sono poi combinati attraverso gli **connettivi logici**.
I test sui valori degli attributi (o termini):

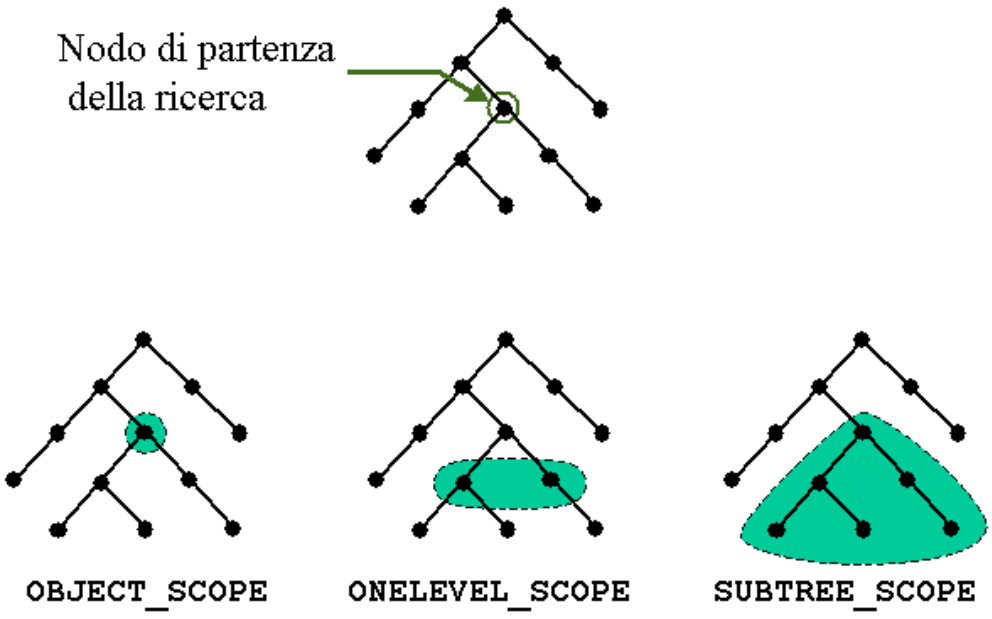
- Uguaglianza:
  - (sn=Roncato) il surname dev'essere Roncato;
- Substring:
  - (sn=*ato) il surname deve terminare in 'ato';
  - (sn=Ron*) il surname deve iniziare con 'Ron';
  - (sn=*ca*) il surname deve contenere 'ca';
  - (sn=R*ca*to) il surname deve iniziare con 'R', finire in 'to' e contenere 'ca';
- Approssimazione:
  - (sn~=Roncato)il surname deve "suonare" come Roncato;
- Confronti di Ordinamento:
  - (sn>=Roncato) il surname dev'essere Roncato o uno che viene dopo nell'ordine alfabetico;
  - (sn<=Roncato) il surname non deve venire dopo Roncato nell'ordine alfabetico;
- Presenza:

- (mail=*) dev’esserci l’attributo mail.

I connettivi logici sono:
AND &
OR |
NOT !
Es. Assenza (!(mail=*)) l’entry non deve avere l’attributo mail

Vincoli

Oltre al filtro, che seleziona quali entry restituire, è possibile specificare anche degli ulteriori vincoli su come effettuare la ricerca e come restituire i risultati.
La funzione poiù importante è quella di specificare lo scope della ricerca:



Ad esempio:
```
SearchControls.OBJECT_SCOPE
SearchControls.ONELEVEL_SCOPE
SearchControls.SUBTREE_SCOPE

vincolo.setSearchScope(...);
```

Per indicare il Search Filter si specifica una combinazione Booleana di test sui valori di attributi:
```
filtro=String();
```

Per indicare gli attributi da restituire:
```
vincoli.setReturningAttributes(String[])
```
Si può anche scegliere di non volere il valore, ma soltanto sapere se gli attributi ci sono:
```
vincoli.setReturningObjFlag(boolean);
```
Limiti: si possono specificare limiti in tempo di ricerca:
```
vincoli.setTimeLimit(int);
```
e in quantità di entry restituite:
```
vincoli.setCountLimit(long);
```

Risultato della ricerca

Vediamo ora come analizzare il risultato di una ricerca. Supponiamo di avere la seguente situazione logica: 3 entry DN1, DN2 e DN3. Ogni entry ha alcuni attributi e qualche attributo ha più valori (es. mail).

- DN1:
  - cn:
    - pippo
  - mail:
    - pippo@localhost
    - pippo@unive.it
- DN2:
  - cn:
    - pluto
  - sn:
    - pluto
- DN3:
  - cn:
    - quak
  - mail:
    - quak@localhost
    - quak@unive.com

Per esaminare il risultato della query abbiamo bisogno quindi di tre cicli:

1. ciclo sulle 0 o più entry, quindi dobbiamo esaminare tutte le entry con un ciclo;
2. ciclo sugli attributi cercati;
3. ciclo sui 0 o più valori degli attributi;

```
NamingEnumeration nomi = ldap.search(base,filtro,vincoli);

while (nomi != null && nomi.hasMore())
{
  SearchResult sr=(SearchResult)nomi.next();

  String dn = sr.getName() + "," + base;

  System.out.println("DN = "+dn);

  String[] nomeAttributi = new String[]{"sn","cn","mail"};

  Attributes ar = ldap.getAttributes(dn, nomeAttributi);

  if (ar != null)
  for (int i =0;i<nomeAttributi.length;i++)
  {
    Attribute attr = ar.get(nomeAttributi[i]);
    if (attr != null)
    {
      System.out.println(nomeAttributi[i]+":");
      Enumeration vals = attr.getAll();

      while (vals.hasMoreElements())
        System.out.println("\t"+vals.nextElement());
```

```
        }
       System.out.println("\n");
      }
    }
```

Per effettuare delle operazioni di modifica dobbiamo effettuare una connessione con Autenticazione e mplementare l'interfaccia DirContext;
Modifica vera e propria che può essere :

- Aggiungere: funzionalità `bind`;
- Modificare: funzionalità `modify`;
- Eliminare: funzionalità `destroy`.

Esempio:

```
java.util.Hashtable env = new
java.util.Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
driver);
env.put(Context.PROVIDER_URL,ldapUrl);

//impostiamo il tipo di autenticazione
env.put(Context.SECURITY_AUTHENTICATION,AUTH_SI
MPLE);

//impostiamo l'utente con cui collegarsi
env.put(Context.SECURITY_PRINCIPAL,user);

//impostiamo la password dell'utente
env.put(Context.SECURITY_CREDENTIALS,password);

ctx = new InitialDirContext(env);
```

Per aggiungere una nuova entry bisogna definire una nuova classe che implementi l'interfaccia DirContext.
Questa classe deve:

- essere in grado di creare il proprio DN;
- sapere come memorizzare i propri attributi;
- e fornire altri meccanismi per gestire i dati recuperati.

Nel nostro esempio vedremo la classe Persona che implementa i seguenti metodi dell'interfaccia DirContext:

- `getAttributes()`;
- il costruttore `Persona()`.

I metodi rimanenti che il compilatore richiede essere definiti affinché la classe implementi l'interfaccia DirContext, sono definiti e generano solamente delle eccezioni. Possiamo farci aiutare da Netbeans o JBuilder per scrivere in automatico tutti questi metodi.

```
public class Person implements DirContext
{
```

```
  String type;
  Attributes myAttrs;

   public Person(String uid,String givenname, String sn, String ou,String
mail)
   {
      myAttrs = new BasicAttributes(true);

      Attribute oc = new BasicAttribute("objectclass");
      oc.add("organizationalPerson");
      oc.add("person");
      oc.add("top");

      Attribute ouSet = new BasicAttribute("ou");
      ouSet.add("People");
      ouSet.add(ou);
      type = uid;

      myAttrs.put(oc);
      myAttrs.put(ouSet);
      myAttrs.put("uid",uid);
      myAttrs.put("sn",sn);
      myAttrs.put("givenname",givenname);
      myAttrs.put("mail",mail);
   }
   public Attributes getAttributes(String name) throws NamingException
   {
      return myAttrs;
   }

   public Attributes getAttributes(Name name) throws NamingException
   {
      return getAttributes(name.toString());
   }

  public Attributes getAttributes(String name, String[] ids) throws
NamingException
  {
   Attributes answer = new BasicAttributes(true);
   Attribute target;
   for (int i = 0; i < ids.length; i++)
   {
     target = myAttrs.get(ids[i]);
     if (target != null)
        answer.put(target);
   }
   return answer;
  }

  public Attributes getAttributes(String name, String[] ids) throws
NamingException
  {
   Attributes answer = new BasicAttributes(true);
   Attribute target;
   for (int i = 0; i < ids.length; i++)
   {
     target = myAttrs.get(ids[i]);
```

```
    if (target != null)
       answer.put(target);
   }
   return answer;
  }

  public Attributes getAttributes(Name name, String[] ids) throws
NamingException
   {
      return getAttributes(name.toString(), ids);
   }

   public String toString()
   {
      return type;
   }
...// tutti gli altri metodi che non fanno niente se non generare
un'eccezione
```

Aggiunta di un Entry

Una volta definita la classe che implementi l'interfaccia DirContext., possiamo procedere ad inserire un oggetto della classe nella directory.
L'inserimento corrisponde a collegare (bind) un nome all'oggetto.
Il nome è ovviamente un DN.
Esempio:

```
Persona p = new Persona("Rossi", "Paolo", "Rossi",
"ou=Accounting","rossi@unive.it");
ctx.bind("uid=prossi,ou=Users,dc=unive,dc=it", p);
```

Modifica di un entry

Modificare un'entry significa:

- aggiungere un attributo;
- eliminare un attributo;
- sostituire il valore di un attributo.

Per effettuare queste operazioni utilizziamo la classe: ModificationItem nel seguente modo:

```
ModificationItem[] mods = new ModificationItem[3];
// vogliamo effettuare tre modifice
// sui seguenti 3 attributi dell'entry:

Attribute a0 =
  new BasicAttribute("telephonenumber", "02-555-2555");

Attribute a1 = new BasicAttribute("sn", "Bianchi");

Attribute a2 =
       new BasicAttribute("mail", "rossi@unive.it");
mods[0] = new ModificationItem(DirContext.ADD_ATTRIBUTE,
a0);
mods[1] = new
```

```
ModificationItem(DirContext.REPLACE_ATTRIBUTE, a1);
mods[2] = new
ModificationItem(DirContext.DELETE_ATTRIBUTE, a2);

ctx.modifyAttributes(MY_ENTRY, mods);
```

Cancellare un entry

Cancellare un entry è l'operazione più semplice: basta indicare il DN della entry da eliminare:

```
ctx.destroySubcontext("uid=prossi,ou=Users,dc=unive,d
c=it");
```

Altre operazioni

`ctx.createSubcontext()`: per aggiungere un'entry;
`ctx.lookup()`: per recuperare una data entry di cui si conosce il DN;
`ctx.rebind()`: per riassegnare il DN ad un altro oggetto/entry;
`ctx.rename()`: per cambiare un DN