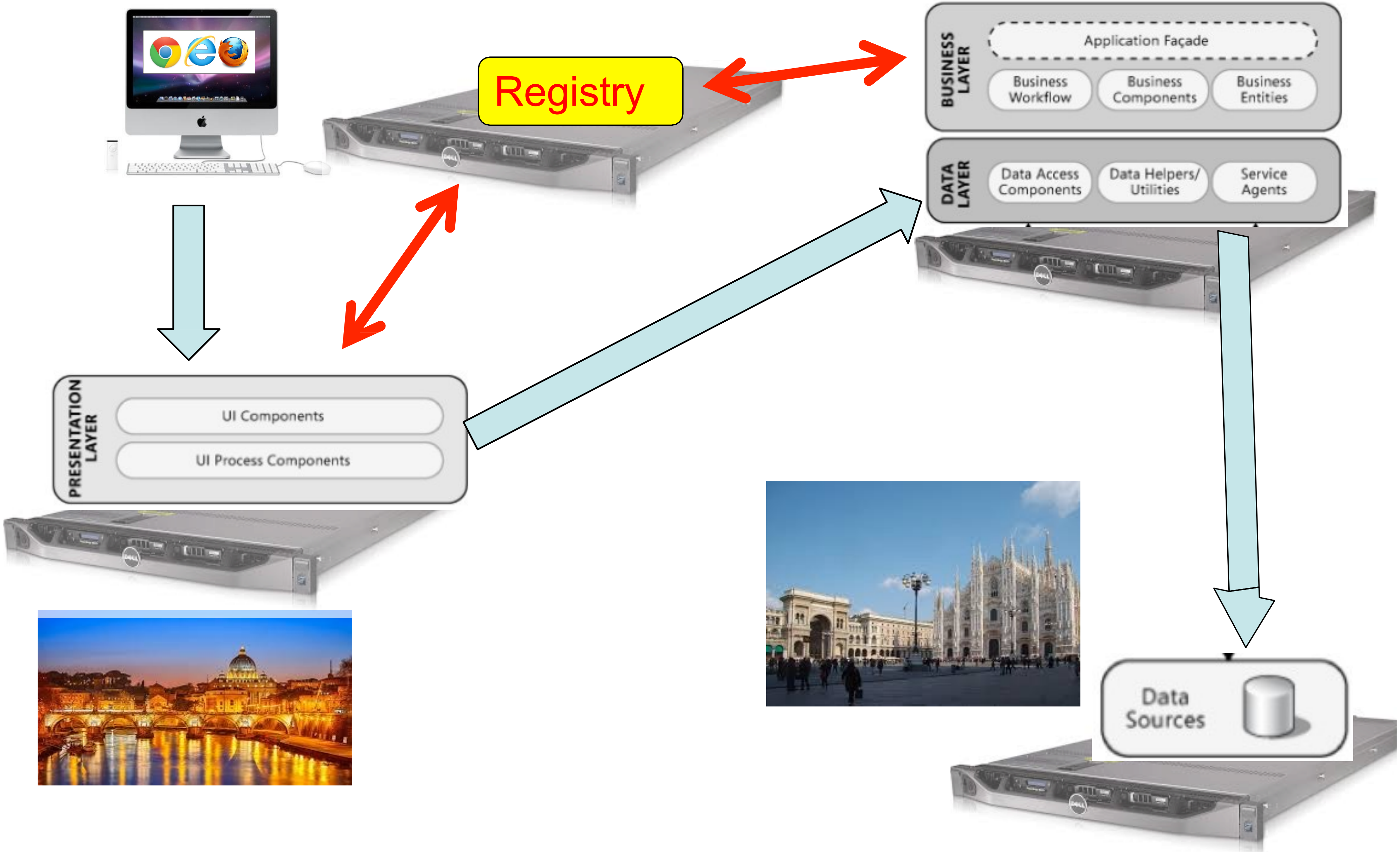
JNDI

Java Naming and Directory Interface

See also:
https://docs.oracle.com/javase/tutorial/jndi/overview/index.html

Distributed Systems



Naming service

A naming service is an entity that

- **associates names with objects.We call this *binding* names to objects.** *This is similar to a telephone company 's associating a person 's name with a specific residence 's telephone number*

- **provides a facility to find an object based on a name.We call this *looking up* or *searching* for an object.***This is similar to a telephone operator finding a person 's telephone number based on that person 's name and connecting the two people.*

In general,a naming service can be used to find any kind of generic object, like a file handle on your hard drive or a printer located across the network.

Directory service

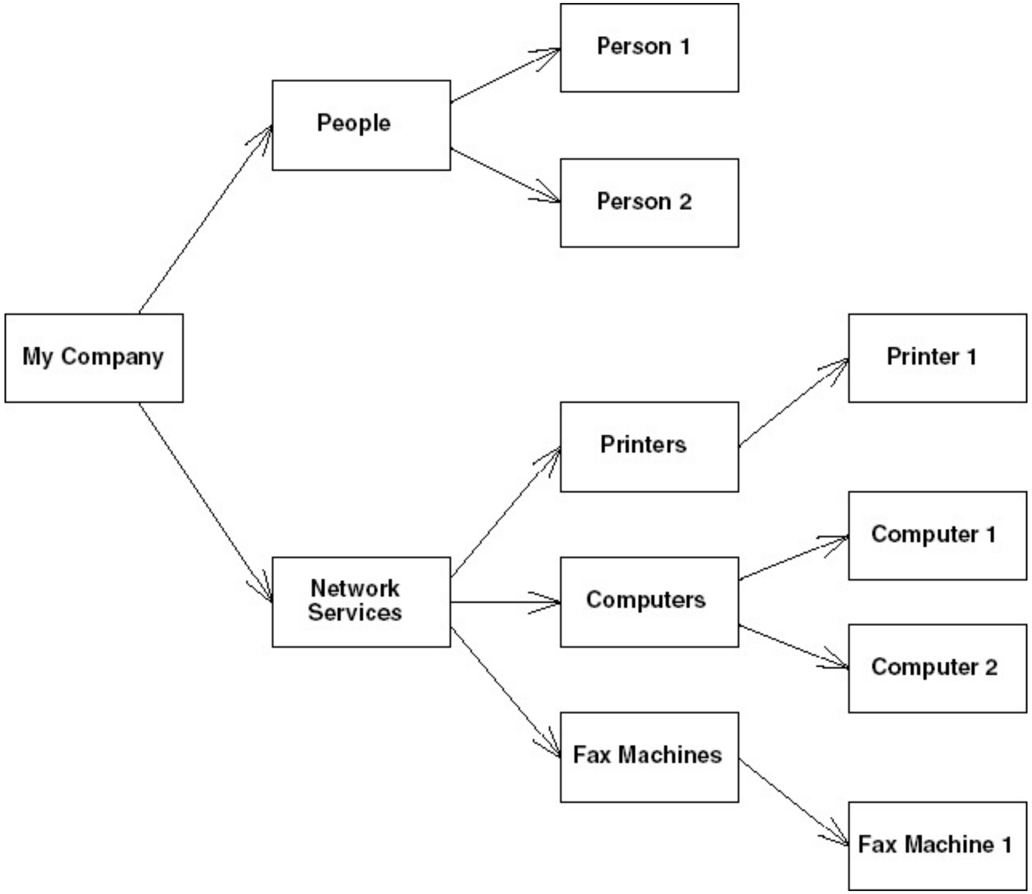
A ***directory object*** **differs from a generic object because you can store *attributes* with directory objects.** *For example, you can use a directory object to represent a user in your company. You can store information about that user, like the user's password, as attributes in the directory object.*

A ***directory service*** **is a naming service that has been extended and enhanced to provide directory object operations for manipulating attributes.**

A ***directory*** **is a system of directory objects that are all connected.** *Some examples of directory products are Netscape Directory Server and Microsoft's Active Directory.*

Directory service

Directories are similar to DataBases, except that they typically are organized in a hierarchical tree-like structure. Typically they are optimized for reading.



Examples of Directory services

Azure Active Directory (Microsoft)

Lotus Domino (IBM)

OpenLDAP (Lightweight Directory Access Protocol)

Apache Directory Service

Red Hat Directory Service (Red Hat)

See also https://en.wikipedia.org/wiki/Directory_service

JNDI concepts

JNDI is a system for Java-based clients to interact with naming and directory systems. JNDI is a bridge over naming and directory services, that provides one common interface to disparate directories.

Users who need to access an LDAP directory use the same API as users who want to access an NIS directory or Novell's directory. All directory operations are done through the JNDI interface, providing a common framework.

JNDI advantages

-You only need to learn a single API to access all sorts of directory service information, such as security credentials, phone numbers, electronic and postal mail addresses, application preferences, network addresses, machine configurations, and more.

-JNDI insulates the application from protocol and implementation details.

-You can use JNDI to read and write whole Java objects from directories.

- You can link different types of directories, such as an LDAP directory with an NDS directory, and have the combination appear to be one large, federated directory.

JNDI advantages

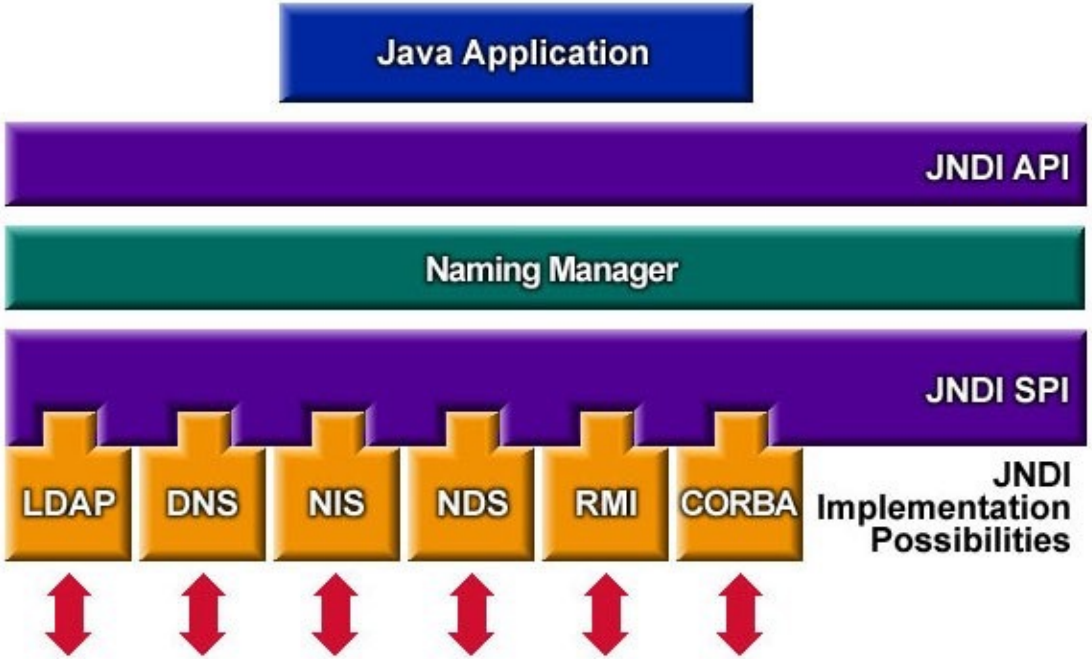
Applications can store factory objects and configuration variables in a global naming tree using the JNDI API.

JNDI, the Java Naming and Directory Interface, provides a global memory tree to store and lookup configuration objects. JNDI will typically contain configured Factory objects.

JNDI lets applications cleanly separate configuration from the implementation. The application will grab the configured factory object using JNDI and use the factory to find and create the resource objects.

In a typical example, the application will grab a database DataSource to create JDBC Connections. Because the configuration is left to the configuration files, it's easy for the application to change databases for different customers.

JNDI Architecture



JNDI concepts

An ***atomic name*** *is a simple,basic,indivisible component of a name.For example,in the string /etc/fstab ,etc and fstab are atomic names.*

A ***binding*** *is an association of a name with an object.*

A ***context*** *is an object that contains zero or more bindings. Each binding has a distinct atomic name. Each of the mtab and exports atomic names is bound to a file on the hard disk.*

A ***compound name*** *is zero or more atomic names put together. e.g. the entire string /etc/fstab is a compound name. Note that a compound name consists of multiple bindings.*

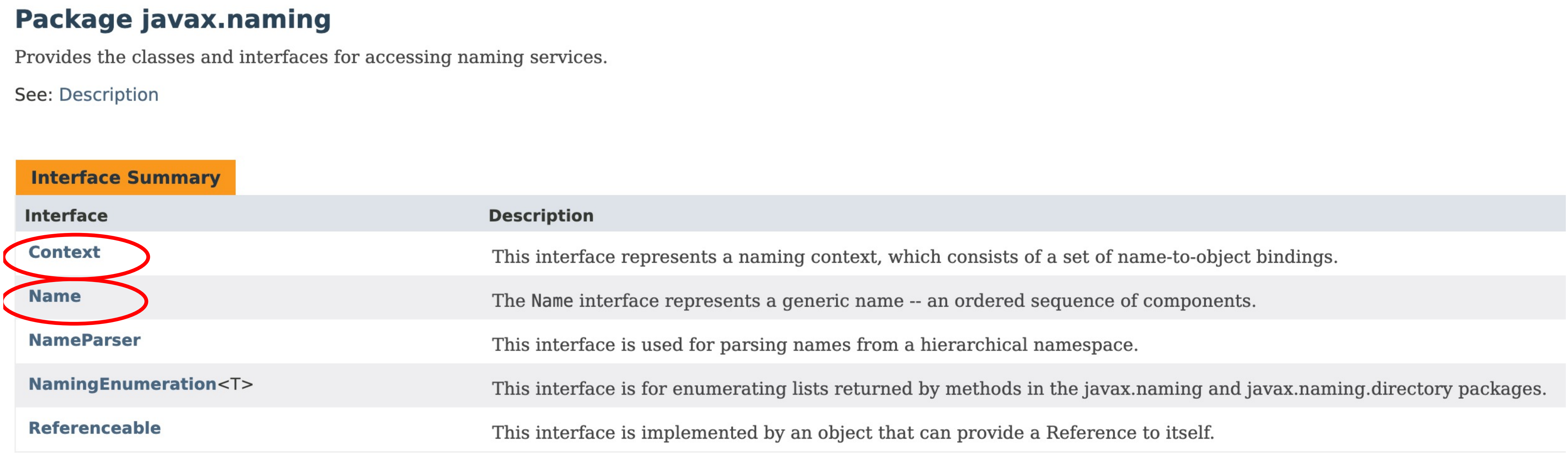
JNDI concepts

A ***reference*** *is a pointer to an object*

A ***naming system*** *is a set of contexts of the same type.*

A ***namespace*** *is the set of all names in a naming system.*

JNDI interfaces



Package javax.naming

Provides the classes and interfaces for accessing naming services.

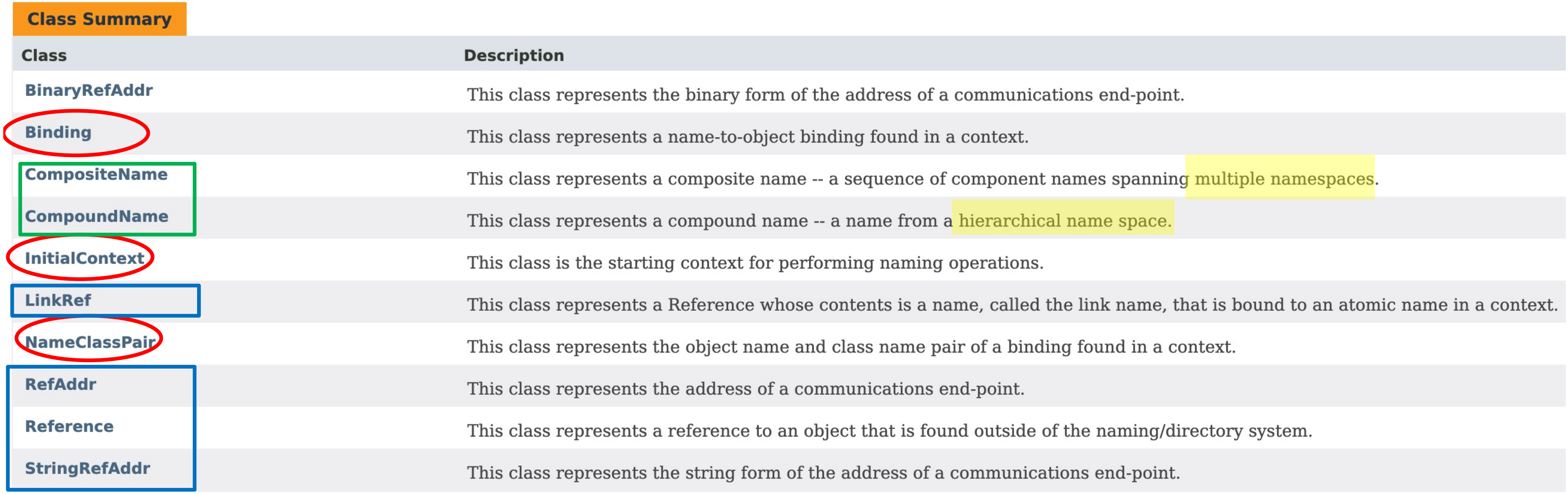
See: Description

Interface Summary

Interface	Description
Context	This interface represents a naming context, which consists of a set of name-to-object bindings.
Name	The `Name` interface represents a generic name -- an ordered sequence of components.
NameParser	This interface is used for parsing names from a hierarchical namespace.
NamingEnumeration<T>	This interface is for enumerating lists returned by methods in the javax.naming and javax.naming.directory packages.
Referenceable	This interface is implemented by an object that can provide a Reference to itself.

https://docs.oracle.com/javase/8/docs/api/javax/naming/package-summary.html

JNDI classes



Class Summary

Class	Description
BinaryRefAddr	This class represents the binary form of the address of a communications end-point.
Binding	This class represents a name-to-object binding found in a context.
CompositeName	This class represents a composite name -- a sequence of component names spanning multiple namespaces.
CompoundName	This class represents a compound name -- a name from a hierarchical name space.
InitialContext	This class is the starting context for performing naming operations.
LinkRef	This class represents a Reference whose contents is a name, called the link name, that is bound to an atomic name in a context.
NameClassPair	This class represents the object name and class name pair of a binding found in a context.
RefAddr	This class represents the address of a communications end-point.
Reference	This class represents a reference to an object that is found outside of the naming/directory system.
StringRefAddr	This class represents the string form of the address of a communications end-point.

https://docs.oracle.com/javase/8/docs/api/javax/naming/package-summary.html

JNDI names

JNDI names look like URLs.
A typical name for a database pool is java:comp/env/jdbc/test.
The java: scheme is a memory-based tree. comp/env is the standard location for Java configuration objects and jdbc is the standard location for database pools.

Examples

java:comp/env	*Configuration environment*
java:comp/env/jdbc	*JDBC DataSource pools*
java:comp/env/ejb	*EJB remote home interfaces*
java:comp/env/cmp	*EJB local home interfaces (non-standard)*
java:comp/env/jms	*JMS connection factories*
java:comp/env/mail	*JavaMail connection factories*
java:comp/env/url	*URL connection factories java:comp/*
UserTransaction	*UserTransaction interface*

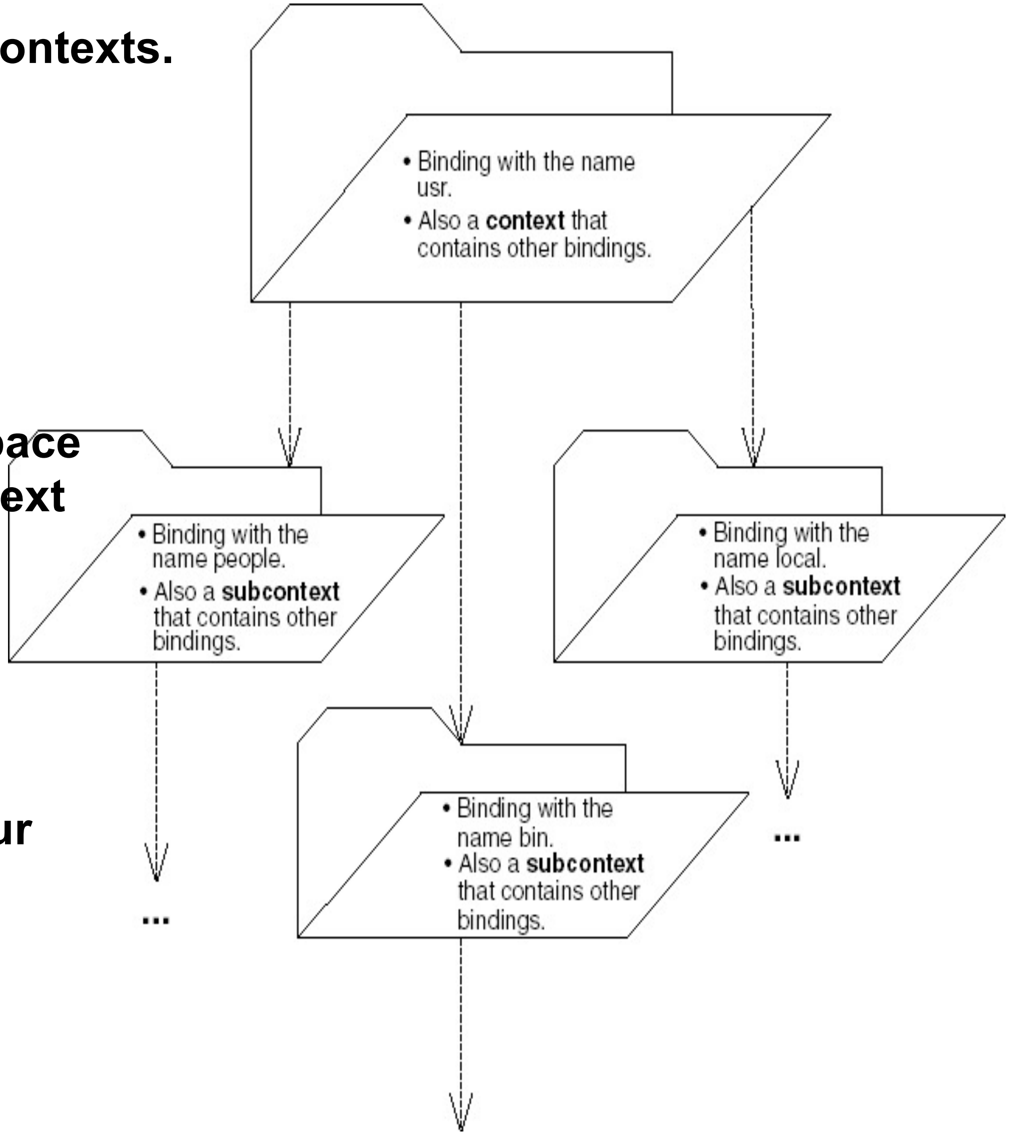
Contexts and Subcontexts

A *naming system* is a connected set of contexts.

A *namespace* is all the names contained within naming system.

The starting point of exploring a namespace is called an *initial context*. An initial context is the first context you happen to use.

To acquire an initial context, you use an *initial context factory*. An initial context factory basically *is* your JNDI driver.



- Binding with the name usr.
- Also a **context** that contains other bindings.
- Binding with the name people.
- Also a **subcontext** that contains other bindings.
- Binding with the name local.
- Also a **subcontext** that contains other bindings.
- Binding with the name bin.
- Also a **subcontext** that contains other bindings.
...
...

Acquiring an initial context

When you acquire an initial context, you must supply the necessary information for JNDI to acquire that initial context.

For example, if you're trying to access a JNDI implementation that runs within a given server, you might supply:

- *The IP address of the server*
- *The port number that the server accepts*
- *The starting location within the JNDI tree*
- *Any username/password necessary to use the server*

Acquiring an initial context

```
package examples;

public class InitCtx {
  public static void main(String args[]) throws Exception {
    // Form an Initial Context
    javax.naming.Context ctx =
          new javax.naming.InitialContext();
    System.err.println("Success!");
    Object result = ctx.lookup("PermissionManager");
  }
}
```

```
java
-Djava.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
-Djava.naming.provider.url=jnp://193.205.194.162:1099
-Djava.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
examples.InitCtx
```

Acquiring an initial context

java.naming.factory.initial: The name of the environment property for specifying the initial context factory to use. The value of the property should be the fully qualified class name of the factory class that will create an initial context.

java.naming.provider.url: The name of the environment property for specifying the location of the service provider the client will use. The NamingContextFactory class uses this information to know which server to connect to. The value of the property should be a URL string

Everything but the host component is optional. The following examples are equivalent because the default port value(on JBOSS) is 4447 (used to be 1099).
remote://www.jboss.org:4447/
www.jboss.org:4447

used to be: jnp://www.jboss.org:1099/

Acquiring an initial context

ja.va.naming.factory.url.pkgs:
The name of the environment property for specifying the list of package prefixes to use when loading in URL context factories. The value of the property should be a colon-separated list of package prefixes for the class name of the factory class that will create a URL context factory.
For the JBoss JNDI provider this must be
org.jboss.ejb.client.naming

(used to be: org.jboss.naming:org.jnp.interfaces).
This property is essential for locating the remote: and java: URL context factories of the JBoss JNDI provider.

Operations on a JNDI context

list() *retrieves a list of contents available at the current context.This typically includes names of objects bound to the JNDI tree,as well as subcontexts.*

listBindings() *retrieves a NamingEnumertion with the names on the current context, and the object associated with them.*

lookup() *moves from one context to another context,such as going from c:\ to c:\windows. You can also use lookup()to look up objects bound to the JNDI tree.The return type of lookup()is JNDI driver specific.*

rename() *gives a context a new name*

Operations on a JNDI context

createSubcontext()*creates a subcontext from the current context,such as creating c:\foo \bar from the folder c:\foo.*

destroySubcontext()*destroys a subcontext from the current context,such as destroying c:\foo \bar from the folder c:\foo.*

bind()*writes something to the JNDI tree at the current context.As with lookup(),JNDI drivers accept different parameters to bind().*

rebind()*is the same operation as bind,except it forces a bind even if there is already something in the JNDI tree with the same name.*

JNDI Examples

Accessing rmiregistry

Using JNDI to access rmiregisty

```
    CompositeName cn=null;
    try { cn = new CompositeName("foo"); }
    catch (InvalidNameException ex) { perr(ex,"Invalid name!"); }
    LinkRef lr=new LinkRef(cn);
    Context ctx=null;
    try { ctx = new InitialDirContext(env);}
    catch (NamingException ex) { perr(ex,"Invalid InitialDirContext!");}
    String name= "myVar3";
    try { Object o=ctx.lookup(name);}
    catch (NamingException ex) {
        System.out.println(name+" is not registered");
        try { ctx.bind(name,lr); }
        catch (NamingException ex1) { perr(ex,"Unable to bind "+name);}
    }
    LinkRef result=null;
    try { result = (LinkRef)ctx.lookup(name) ;}
    catch (NamingException ex) { perr(ex,"Unable to lookup "+name);}
    try { System.out.println(result.getLinkName()); }
    catch (NamingException ex) {perr(ex,"Unable to get name from LinkRef ");}
    try { ctx.close();}
    catch (NamingException ex) {perr(ex,"Error on close");}
}}
```

create the object to be stored: in this case a (storable) type of String

acquire the context

if the name is not yet registered, register it

look up the name

print its value

close the connection

*Code is on the course web site - **REMEMBER TO START rmiregistry first!***

Using JNDI to access rmiregisty

see http://docs.oracle.com/javase/8/docs/technotes/guides/jndi/jndi-rmi.html

```
package jndiaccesstormiregistry;

import java.util.Properties;
import javax.naming.CompositeName;
import javax.naming.Context;
import javax.naming.InvalidNameException;
import javax.naming.LinkRef;
import javax.naming.NamingException;
import javax.naming.directory.InitialDirContext;

public class Demo {

    public static void main(String[] args) {
        // Identify service provider to use
        Properties env = new Properties();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.jndi.rmi.registry.RegistryContextFactory");
        env.put(Context.PROVIDER_URL, "rmi://localhost:1099");
```

```
private static void perr(Exception ex, String message) {
        System.out.println(message);
        ex.printStackTrace();
        System.exit(1);
    }
```

Using JNDI to access rmiregisty

NOTE: we are forcing rmiregistry to do something it wasn't designed for (storing strings)

rmiregistry is FLAT – no subcontexts!

An old but interesting additional reading about rmiregistry:

http://www.drdobbs.com/jvm/a-remote-java-rmi-registry/212001090?pgno=1

JNDI Examples

Accessing LDAP

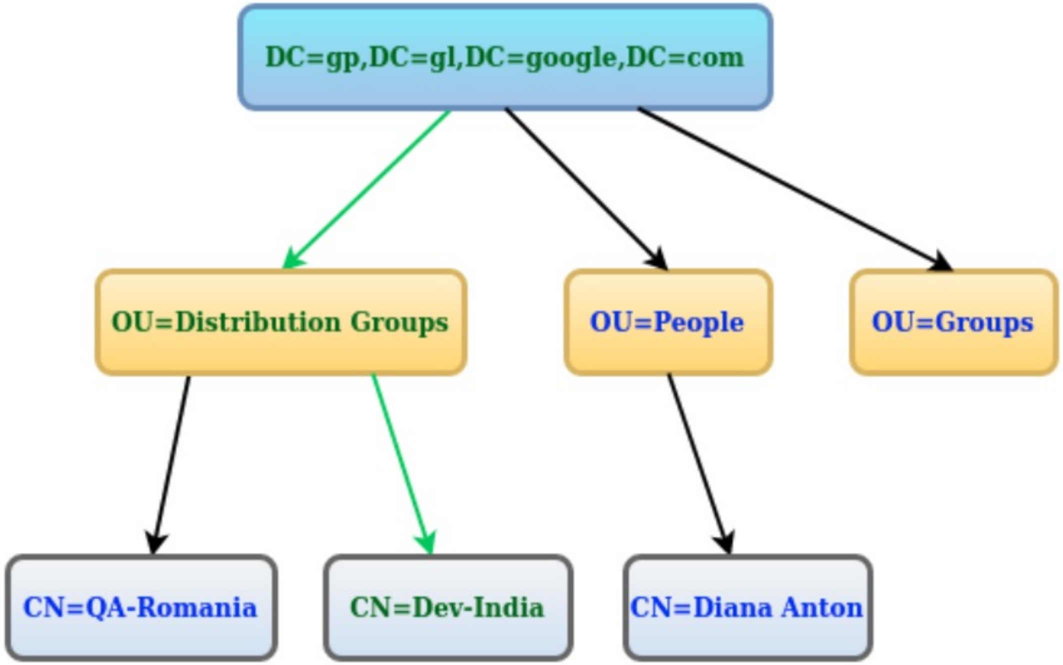
LDAP Namespace Structure

Information in an *LDAP* database comes in the form of objects.
Objects have attributes that describe them.

When an *LDAP* client needs to locate information about an object, it submits a query that contains the object's distinguished name (DN) and the attributes the client wants to see.

CN=Dev-Romania OU=Distribution Groups DC=gp, DC=gl, DC=google, DC=com

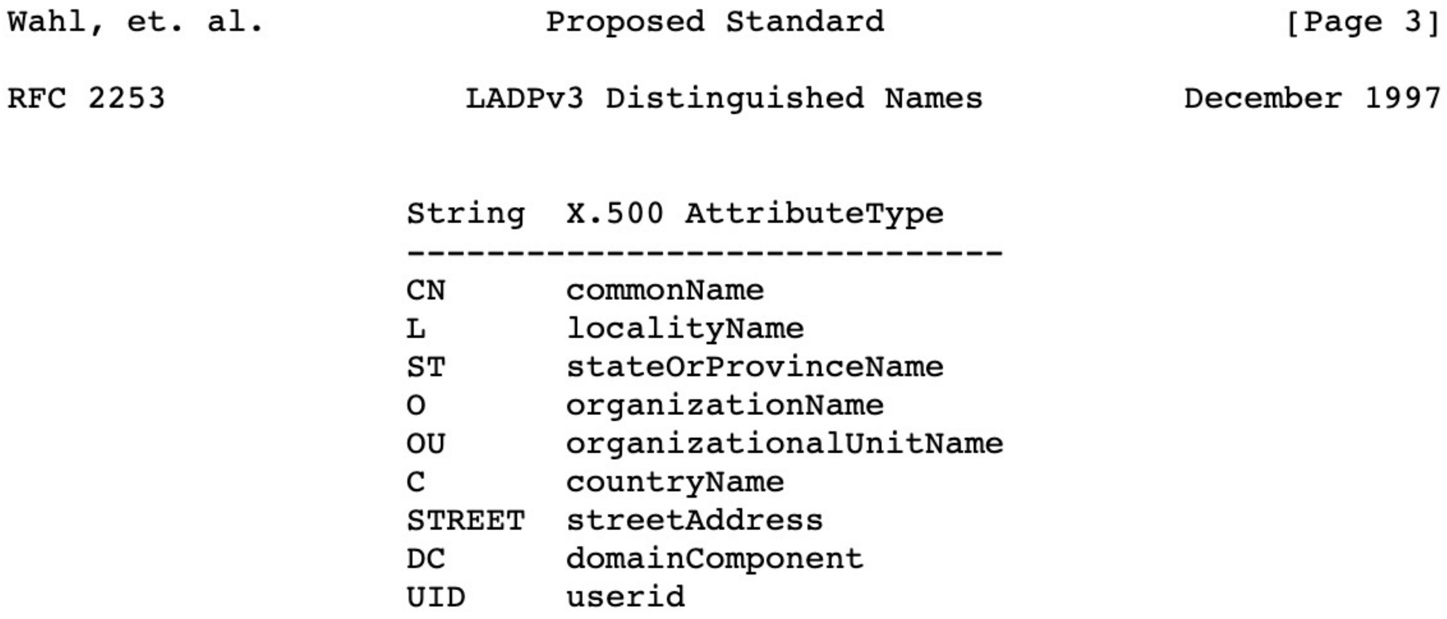
Read right to left



LDAP Namespace Structure

A name that includes an object's entire path to the root of the *LDAP* namespace is called its ***distinguished name***, or DN.

An object name without a path, or a partial path, is called a ***relative distinguished name***, or RDN.

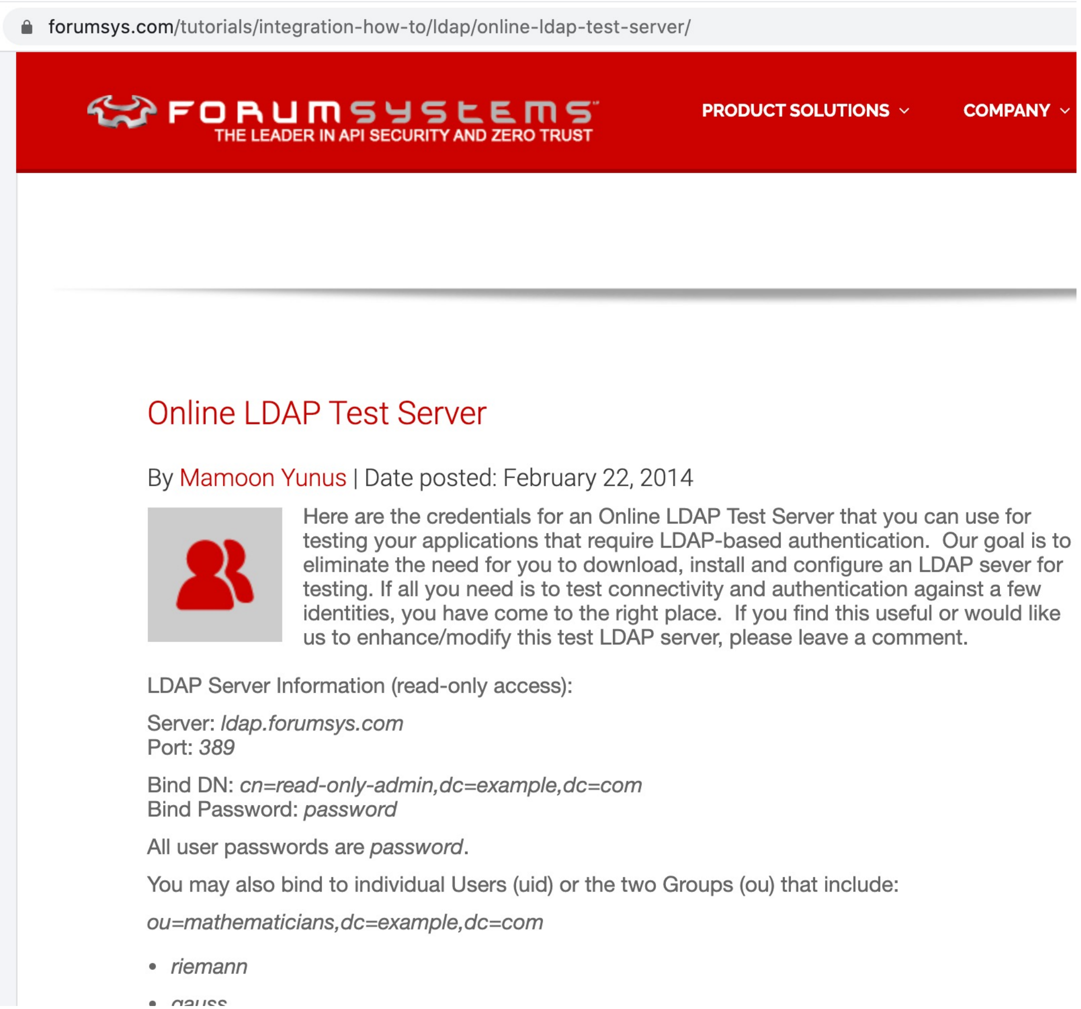


Wahl, et. al.   Proposed Standard   [Page 3]

RFC 2253   LADPv3 Distinguished Names   December 1997

String	X.500 AttributeType
CN	commonName
L	localityName
ST	stateOrProvinceName
O	organizationName
OU	organizationalUnitName
C	countryName
STREET	streetAddress
DC	domainComponent
UID	userid

If you want to practice…


forumsys.com/tutorials/integration-how-to/ldap/online-ldap-test-server/
FORUMSYSTEMS
THE LEADER IN API SECURITY AND ZERO TRUST
PRODUCT SOLUTIONS
COMPANY

Online LDAP Test Server

By Mamoon Yunus | Date posted: February 22, 2014

Here are the credentials for an Online LDAP Test Server that you can use for testing your applications that require LDAP-based authentication. Our goal is to eliminate the need for you to download, install and configure an LDAP sever for testing. If all you need is to test connectivity and authentication against a few identities, you have come to the right place. If you find this useful or would like us to enhance/modify this test LDAP server, please leave a comment.

LDAP Server Information (read-only access):

Server: *ldap.forumsys.com*
Port: *389*

Bind DN: *cn=read-only-admin,dc=example,dc=com*
Bind Password: *password*

All user passwords are *password*.

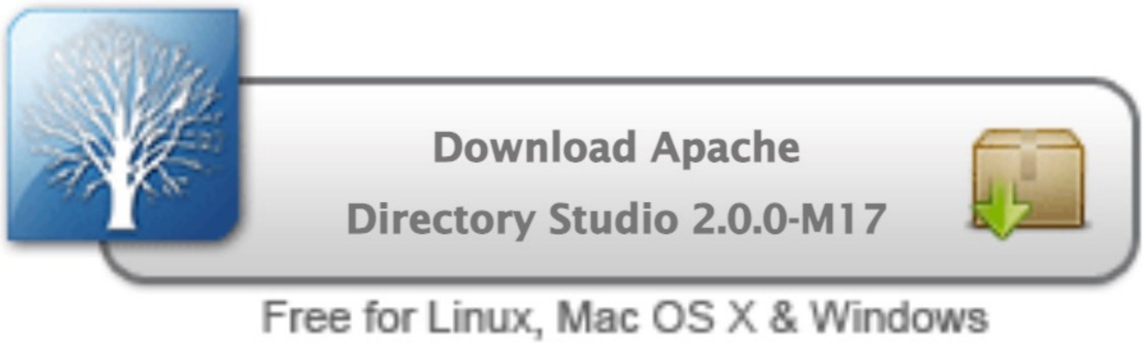
You may also bind to individual Users (uid) or the two Groups (ou) that include:

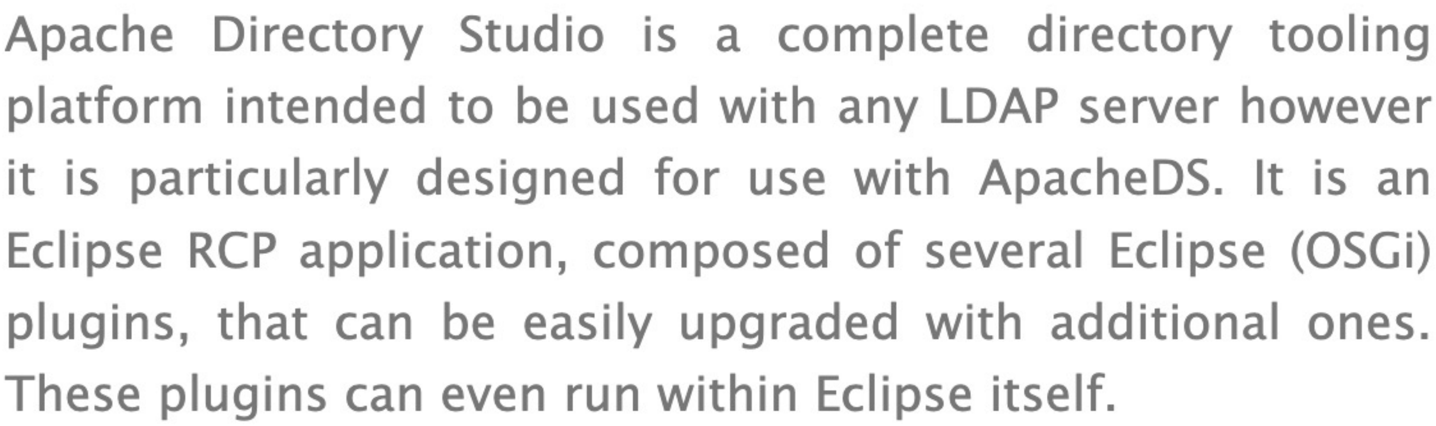
ou=mathematicians,dc=example,dc=com

- *riemann*
- *gauss*

https://www.forumsys.com/tutorials/integration-how-to/ldap/online-ldap-test-server/

If you want to practice…


Download Apache
Directory Studio 2.0.0-M17
Free for Linux, Mac OS X & Windows


Apache Directory Studio is a complete directory tooling platform intended to be used with any LDAP server however it is particularly designed for use with ApacheDS. It is an Eclipse RCP application, composed of several Eclipse (OSGi) plugins, that can be easily upgraded with additional ones. These plugins can even run within Eclipse itself.

https://directory.apache.org/

A JNDI-LDAP example

```
public class InitCtx {
   public InitCtx() {
      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
      env.put(Context.PROVIDER_URL, = "ldap://ldap.forumsys.com:389");
      //env.put(Context.SECURITY_AUTHENTICATION, "simple");
      //env.put(Context.SECURITY_PRINCIPAL, "cn=read-only-admin,dc=example,dc=com");
      //env.put(Context.SECURITY_CREDENTIALS, "password");
      DirContext ctx = null;
      try {
         ctx = new InitialDirContext(env);
         System.out.println("connected");
         System.out.println(ctx.getEnvironment());
         // do something useful with the context...
         NamingEnumeration<NameClassPair> list
              = ctx.list("dc=example,dc=com");
         printList(list);
         Attributes attrs=ctx.getAttributes(
                    "uid=curie,dc=example,dc=com");
         System.out.println("sn: " +
                     attrs.get("cn").get());
         ctx.close();

      } catch (NamingException ex) {
         ex.printStackTrace();
      }
   }
```

```
void printList(NamingEnumeration<NameClassPair> list) {
      try {
         while (list.hasMore()) {
            NameClassPair ncp = list.next();
            System.out.println(ncp.getClassName() +
              " " + ncp.getName() + " " +
             ncp.getNameInNamespace());
         }
      } catch (NamingException e) {
         e.printStackTrace();
      }
   }
```

```
      public static void main(String[] a) { new InitCtx();  }
}
```

A JNDI-LDAP example

```
public class InitCtx {
  public InitCtx() {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, = "ldap://ldap.forumsys.com:389");
    //env.put(Context.SECURITY_AUTHENTICATION, "simple");
    //env.put(Context.SECURITY_PRINCIPAL, "cn=read-only-admin,dc=example,dc=com");
    //env.put(Context.SECURITY_CREDENTIALS, "password");
    DirContext ctx = null;
    try {
      ctx = new InitialDirContext(env);
      System.out.println("connected");
      System.out.println(ctx.getEnvironment());
      // do something useful with the context...
      NamingEnumeration<NameClassPair> list
          = ctx.list("dc=example,dc=com");
      printList(list);
      Attributes attrs=ctx.getAttributes(
              "uid=curie,dc=example,dc=com");
      System.out.println("sn: " +
               attrs.get("cn").get());
      ctx.close();

    } catch (NamingException ex) {
      ex.printStackTrace();
    }
}
```

Alternative writing

```
DirContext ctx1 = (DirContext) ctx.lookup("dc=example,dc=com");
Attributes attrs=ctx1.getAttributes("uid=curie");
```

Code is on the course web site – ***DOES NOT WORK FROM WITHIN UNITN (firewall)!***

JBOSS/Wildfly

What is JBoss/Wildfly ?

- JBoss Application Server (or JBoss AS) is a free software/open-source Java EE-based application server.
- Not only implements a server that runs on Java, but it actually implements the Java EE part of Java.
- Because it is Java-based, the JBoss application server operates cross-platform: usable on any operating system that supports Java.
- JBoss AS was developed by JBoss, now a division of Red Hat.

JEE

- JEE provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.
- Java EE extends the Java SE providing API for object-relational mapping, distributed and multi-tier architectures, and web services.
- The platform incorporates a design based largely on modular components running on an application server.

Key JEE Components

- **EJB** (Enterprise JavaBeans) define a distributed component system that is the heart of the Java EE specification . This system , in fact, provides the typical features required by enterprise applications , such as scalability, security , data persistence , and more.
- **JNDI** defines a system to identify and list generic resources , such as software components or data sources .
- **JDBC** is an interface for access to any type of data bases.
- **JTA** is a system for distributed transaction support .
- **JPA** is an API for the management of persistent data .
- **JAXP** is an API for handling files in XML format.
- **JMS** (Java Message Service) a system for sending and managing messages.

Installing Wildfly

- http://wildfly.org/downloads/

- Download 20.0.1.Final

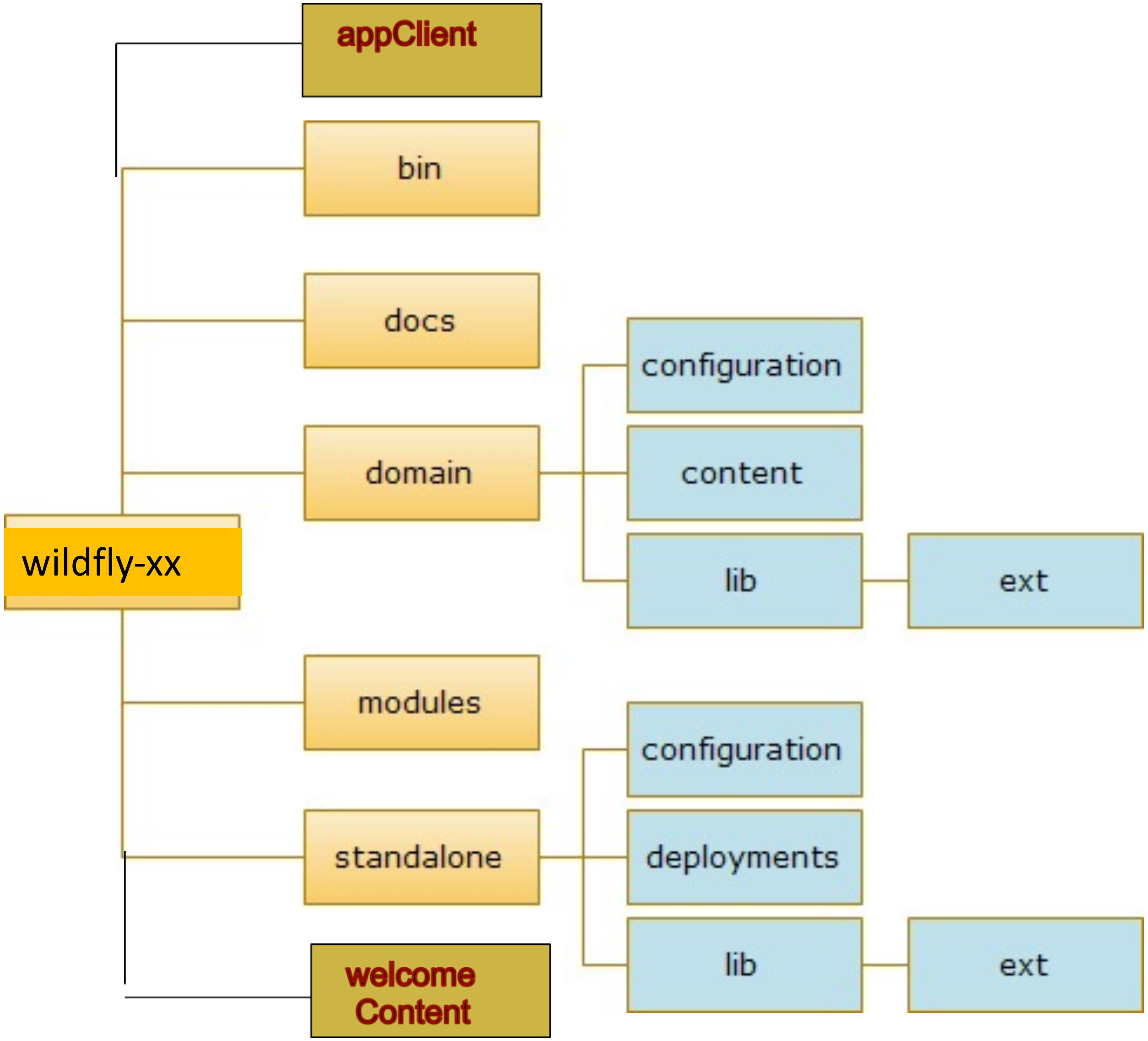
.

GETTING STARTED GUIDE:
http://docs.wildfly.org/20/

- unzip it where you like: that will be your JBOSS_HOME
- define it as a login variable, e.g. in bash

export JBOSS_HOME=*yourpath*

Wildfly directory structure



See http://docs.wildfly.org/20/Getting_Started_Guide.html#wildfly---a-quick-tour

Starting and stopping Wildfly

cd into your JBOSS_HOME/bin

UNIX (LINUX-MAC)

- START: `./standalone.sh &`
- STOP: `./jboss-cli.sh --connect command=:shutdown`

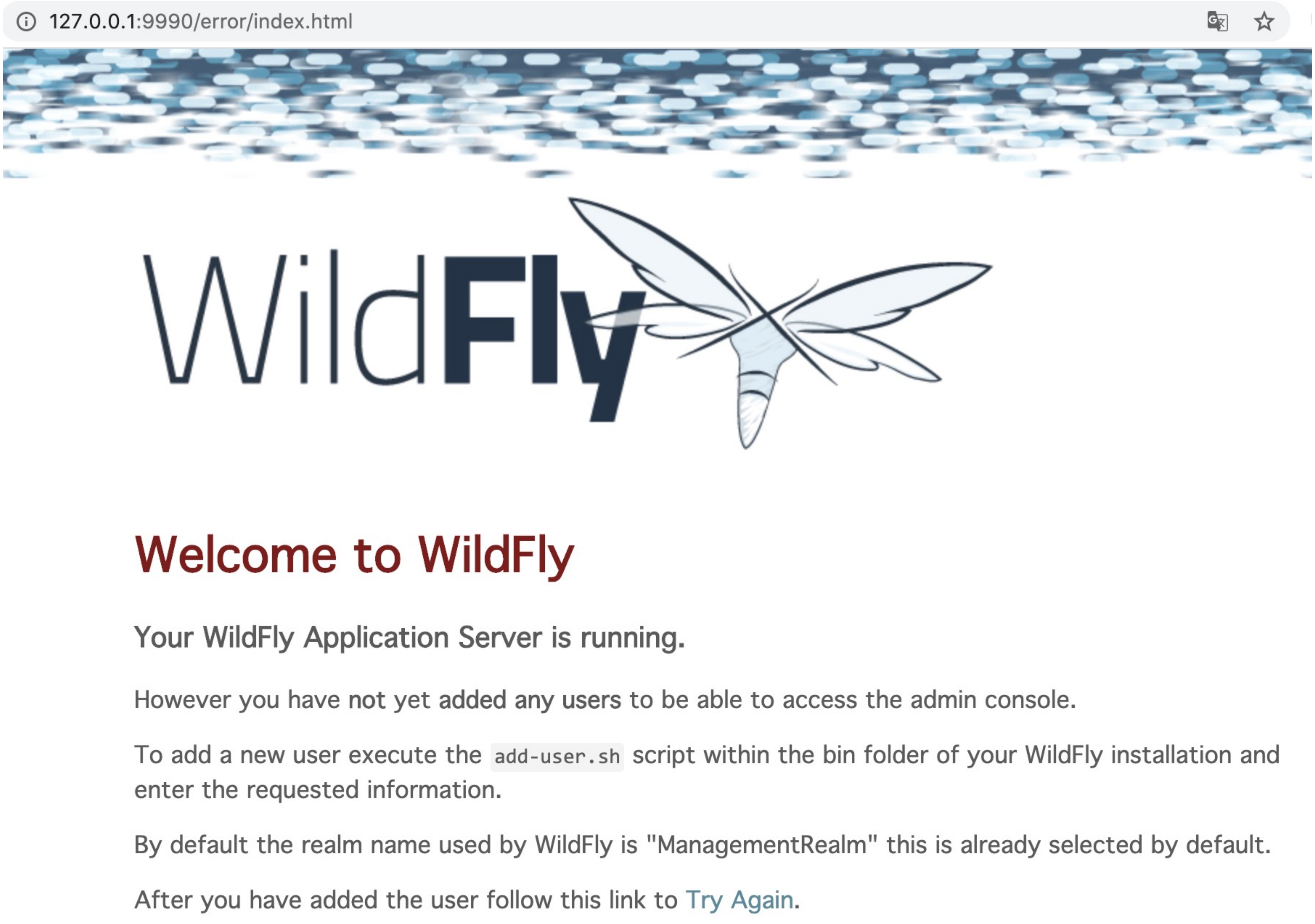
WINDOWS

- START: `./standalone.bat`
- STOP: `./jboss-cli.bat --connect command=:shutdown`

On starting….

```
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /Users/ronchet/Downloads/wildfly-21.0.0.Final

  JAVA: /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/java

  JAVA_OPTS:  -server -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m
-Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman
-Djava.awt.headless=true

=========================================================================

17:20:18,631 INFO  [org.jboss.modules] (main) JBoss Modules version 1.10.2.Final
17:20:19,035 INFO  [org.jboss.msc] (main) JBoss MSC version 1.4.12.Final
17:20:19,042 INFO  [org.jboss.threads] (main) JBoss Threads version 2.4.0.Final
17:20:19,129 INFO  [org.jboss.as] (MSC service thread 1-2) WFLYSRV0049: WildFly Full
21.0.0.Final (WildFly Core 13.0.1.Final) starting
17:20:19,567 INFO  [org.wildfly.security] (ServerService Thread Pool -- 27)
    ELY00001: WildFly Elytron version 1.13.1.Final
…
17:20:20,491 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0025:
WildFly Full 21.0.0.Final (WildFly Core 13.0.1.Final) started in 2213ms - Started 317 of 579 ser
17:20:20,492 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0060:
    Http management interface listening on http://127.0.0.1:9990/management
17:20:20,492 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0051:
    Admin console listening on http://127.0.0.1:9990
```

watch for errors!



Welcome to WildFly

Your WildFly Application Server is running.

However you have **not** yet **added any users** to be able to access the admin console.

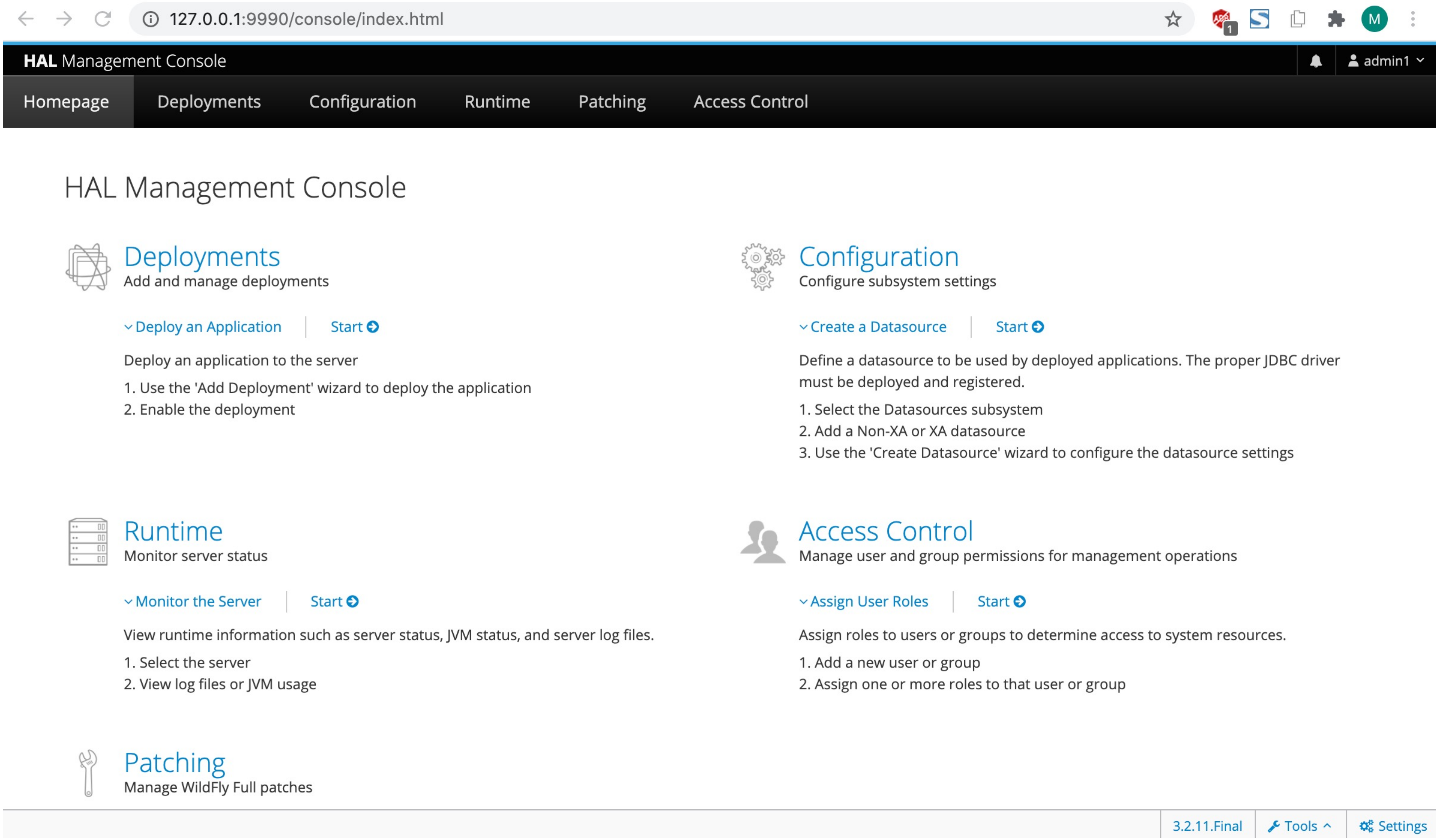
To add a new user execute the `add-user.sh` script within the bin folder of your WildFly installation and enter the requested information.

By default the realm name used by WildFly is "ManagementRealm" this is already selected by default.

After you have added the user follow this link to Try Again.

Wildfly: Creating an admin user

```
ronchet$ ./add-user.sh
What type of user do you wish to add?
a)   Management User (mgmt-users.properties)
b)    Application User (application-users.properties)
(a): a
Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : admin123
Password recommendations are listed below. … The password should contain at least 8
characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password : password1$
 Re-enter Password : password1$
What groups do you want this user to belong to? (Please enter a comma separated list, or
leave blank for none)[ ]:  <return>
About to add user 'admin123' for realm 'ManagementRealm'
 Is this correct yes/no? yes
 Added user 'admin123' to file '/Users/…/jboss/wildfly-
16.0.0.Final/standalone/configuration/mgmt-users.properties'
Added user 'admin123…
Is this new user going to be used for one AS process to connect to another AS process? e.g.
for a slave host controller connecting to the master or for a Remoting connection for server
to server EJB calls. yes/no? yes
…
```



HAL Management Console

Deployments

Add and manage deployments

Deploy an Application | Start

Deploy an application to the server

1. Use the 'Add Deployment' wizard to deploy the application
2. Enable the deployment

Configuration

Configure subsystem settings

Create a Datasource | Start

Define a datasource to be used by deployed applications. The proper JDBC driver must be deployed and registered.

1. Select the Datasources subsystem
2. Add a Non-XA or XA datasource
3. Use the 'Create Datasource' wizard to configure the datasource settings

Runtime

Monitor server status

Monitor the Server | Start

View runtime information such as server status, JVM status, and server log files.

1. Select the server
2. View log files or JVM usage

Access Control

Manage user and group permissions for management operations

Assign User Roles | Start

Assign roles to users or groups to determine access to system resources.

1. Add a new user or group
2. Assign one or more roles to that user or group

Patching

Manage WildFly Full patches

making WildFly accessible from remote

edit the server descriptor:

- cd into $JBOSS_HOME/standalone/configuration/
- edit standalone.xml

toward the end of the file, find the interfaces definitions:

<interfaces>

…

<interfaces/>

ADD A NEW INTERFACE (give it a name: in this example *myInterface*), setting the access of it to any ip.

security
problem!

making WildFly accessible from remote

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}">
    </inet-address></interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}">
    </inet-address></interface>
    <interface name="unsecure">
        <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}">
    </inet-address></interface>
    <interface name="myInterface">
        <any-address/>
    </interface>
  </interfaces>
```

making JBOSS accessible from remote

find the socket-binding-group tag, which sets the ports required for the given interfaces.

change the default-interface parameter to match the new interface (*myInterface* in ourcase)

was:

```
<socket-binding-group default-interface="default" name="standard-sockets" port-offset="${jboss.socket.binding.port-offset:0}">
```

must become:

```
<socket-binding-group default-interface="myInterface" name="standard-sockets" port-offset="${jboss.socket.binding.port-offset:0}">
```

Save, and restart the server

JBOSS: Creating other users

```
What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): b

Enter the details of the new user to add.
Using realm 'ApplicationRealm' as discovered from the existing property files.
Username : user
Password recommendations are listed below. …The password should contain at least 8
characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password : password1$
Re-enter Password :  password1$
What groups do you want this user to belong to? (Please enter a comma separated list, or
leave blank for none)[  ]:  <return>
About to add user 'user' for realm 'ApplicationRealm'
Is this correct yes/no? yes
Added user 'user' to file ...
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for
server to server EJB calls.
yes/no? yes
```

Q

How can we use Wildfly for JNDI ?

Adding JNDI bindings

1) locate in your standalone/configuration/standalone.xml the line

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
       <remote-naming/>
  </subsystem>
```

2) replace it with the following section

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
   <remote-naming/>
   <bindings>
        <simple name="java:global/a" value="100" type="int" />
        <simple name="java:global/myhome" value="https://latemar.science.unitn.it"
type="java.net.URL" />
        <simple name="java:jboss/exported/jndi/mykey" value="MyJndiValue"/>
        <lookup name="java:jboss/exported/link/mykey"
lookup="java:jboss/exported/jndi/mykey"/>
   </bindings>
</subsystem>
```

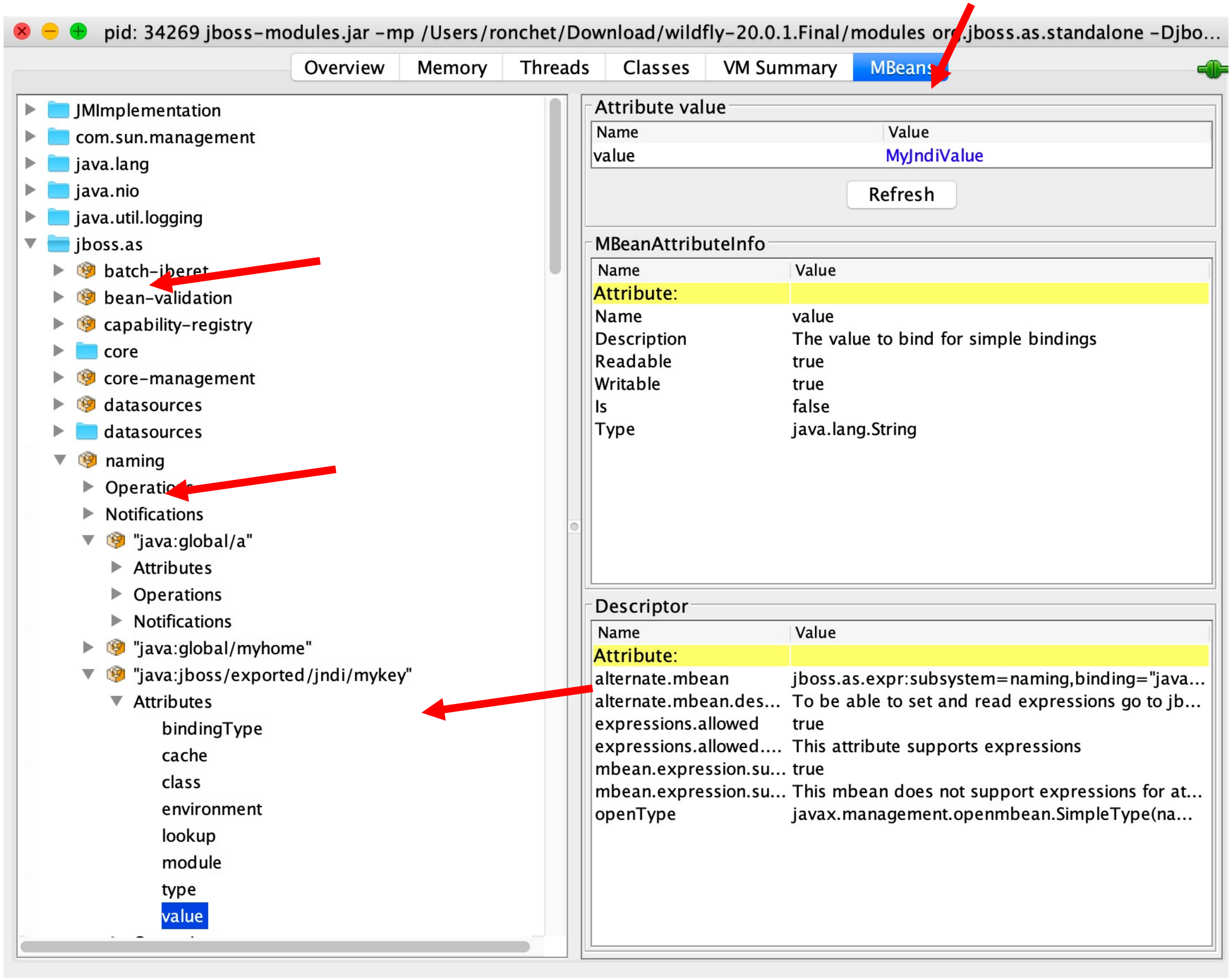
3) restart your server

NOTE: the space visible on the client is the one following java:jboss/exported/

On server: java:jboss/exported/jndi/mykey On client: jndi/mykey

Inspecting your server

1. **start jconsole (It's in the java distribution, bin directory)**
2. **connect with your wildfly running instance**
3. **inspect the jndi bindings (jboss.as -> naming)**



pid: 34269 jboss-modules.jar -mp /Users/ronchet/Download/wildfly-20.0.1.Final/modules org.jboss.as.standalone -Djbo...

- JMImplementation
- com.sun.management
- java.lang
- java.nio
- java.util.logging
- jboss.as
  - batch-jberet
  - bean-validation
  - capability-registry
  - core
  - core-management
  - datasources
  - datasources
  - naming
    - Operations
    - Notifications
    - "java:global/a"
      - Attributes
      - Operations
      - Notifications
    - "java:global/myhome"
    - "java:jboss/exported/jndi/mykey"
      - Attributes
        - bindingType
        - cache
        - class
        - environment
        - lookup
        - module
        - type
        - value

Attribute value

Name	Value
value	MyJndiValue

MBeanAttributeInfo

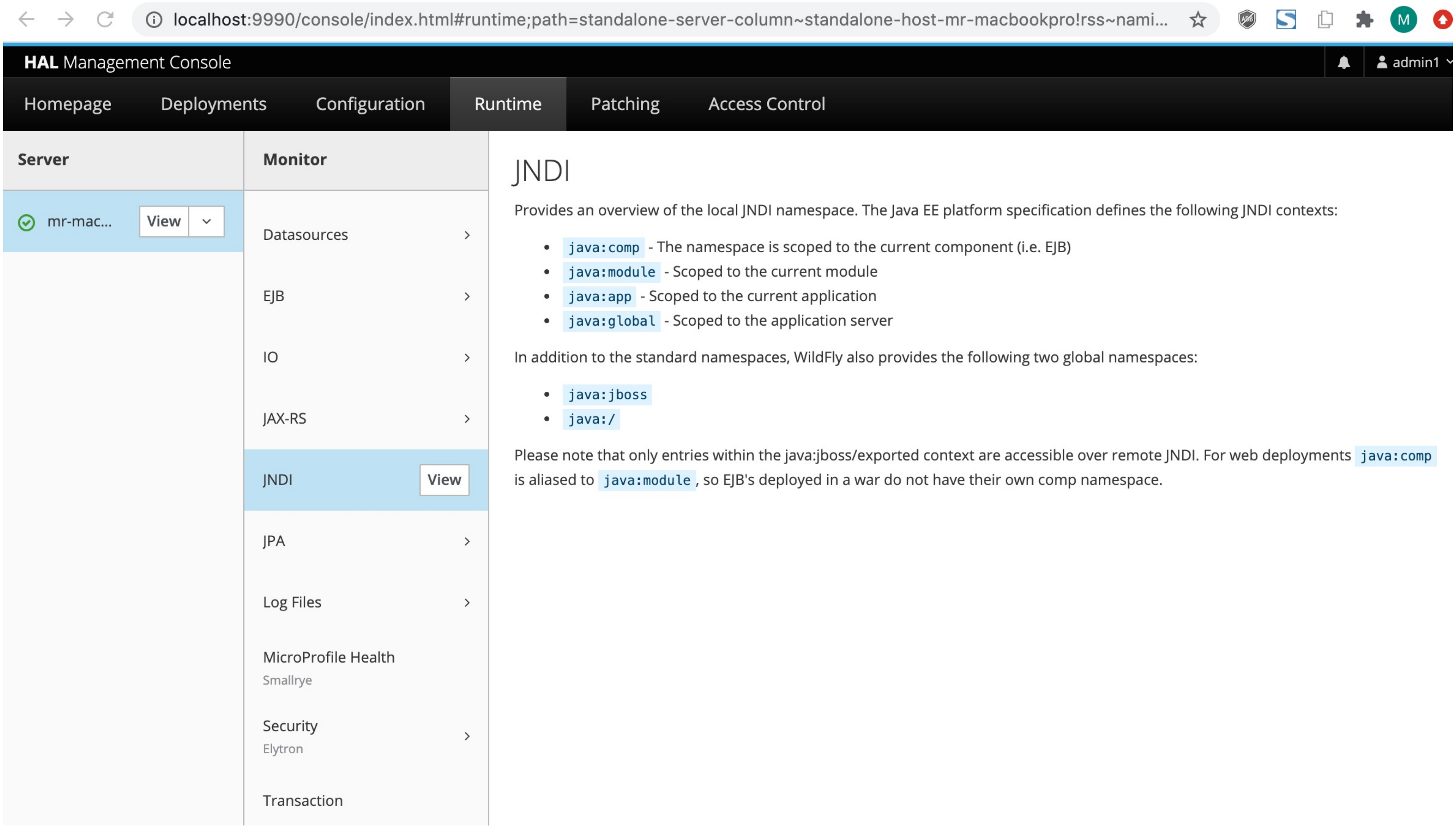
Name	Value
Attribute:	
Name	value
Description	The value to bind for simple bindings
Readable	true
Writable	true
Is	false
Type	java.lang.String

Descriptor

Name	Value
Attribute:	
alternate.mbean	jboss.as.expr:subsystem=naming,binding="java...
alternate.mbean.des...	To be able to set and read expressions go to jb...
expressions.allowed	true
expressions.allowed....	This attribute supports expressions
mbean.expression.su...	true
mbean.expression.su...	This mbean does not support expressions for at...
openType	javax.management.openmbean.SimpleType(na...

Inspecting your server

1. **web console ->runtime -> start**



JNDI

Provides an overview of the local JNDI namespace. The Java EE platform specification defines the following JNDI contexts:

- `java:comp` - The namespace is scoped to the current component (i.e. EJB)
- `java:module` - Scoped to the current module
- `java:app` - Scoped to the current application
- `java:global` - Scoped to the application server

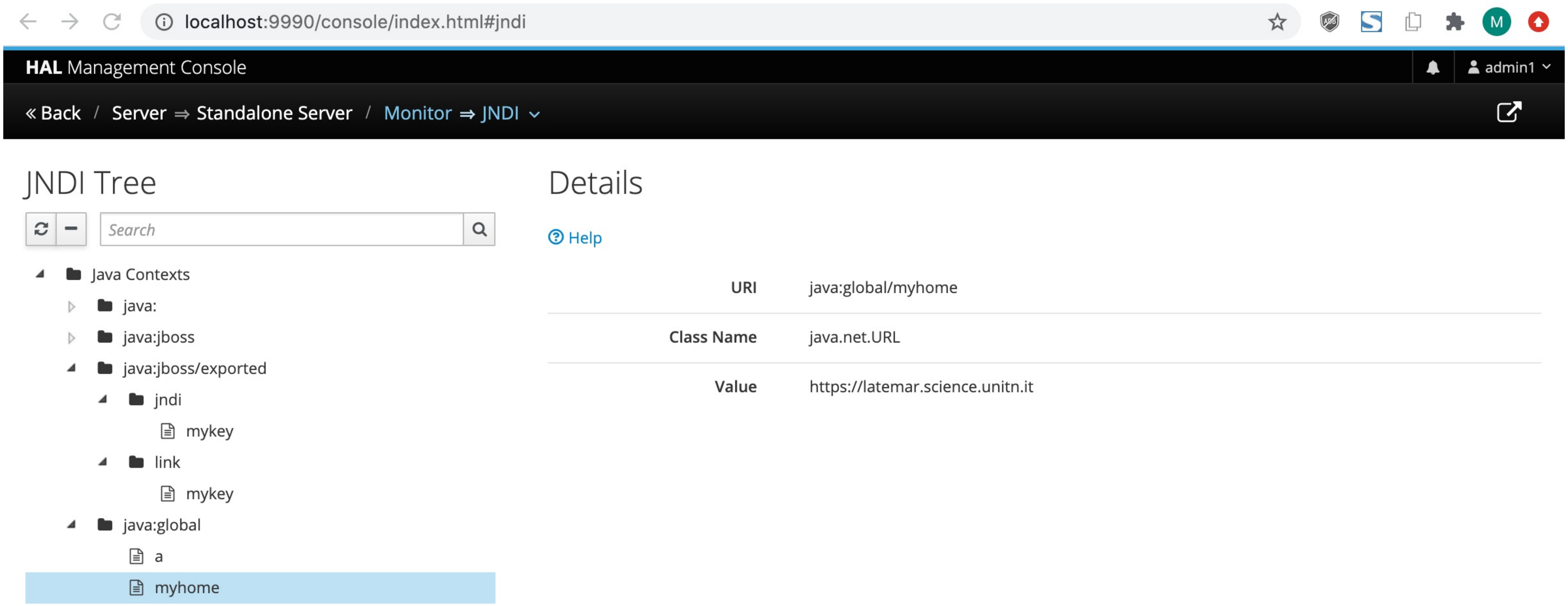
In addition to the standard namespaces, WildFly also provides the following two global namespaces:

- `java:jboss`
- `java:/`

Please note that only entries within the java:jboss/exported context are accessible over remote JNDI. For web deployments `java:comp` is aliased to `java:module`, so EJB's deployed in a war do not have their own comp namespace.

Inspecting your server

2. View



localhost:9990/console/index.html#jndi

JNDI Tree

- Java Contexts
  - java:
  - java:jboss
  - java:jboss/exported
    - jndi
      - mykey
    - link
      - mykey
  - java:global
    - a
    - myhome

Details

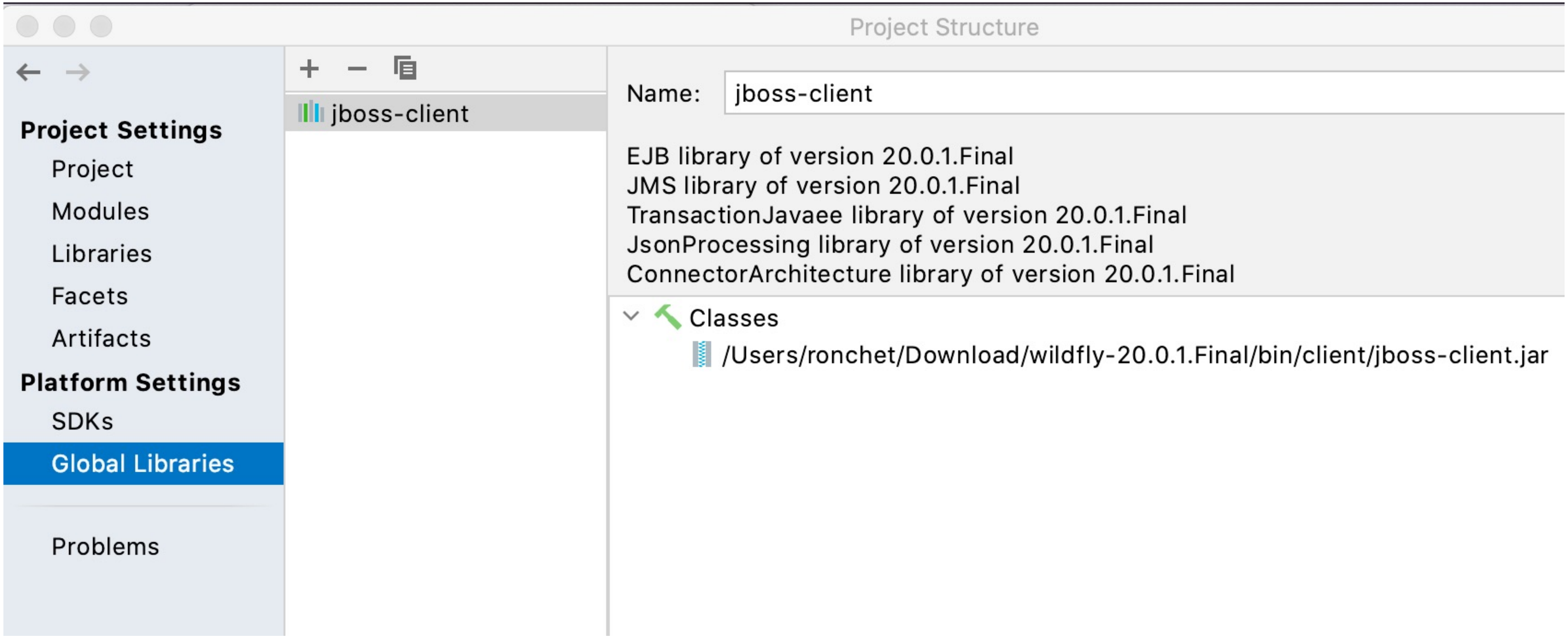
URI	java:global/myhome
Class Name	java.net.URL
Value	https://latemar.science.unitn.it

Accessing via code

```
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class JNDIaccess {
    public static void main(String a[]) throws NamingException {
        new JNDIaccess();
    }
    public JNDIaccess() throws NamingException {
        Properties jndiProps = new Properties();
        jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,
                "org.wildfly.naming.client.WildFlyInitialContextFactory");
        jndiProps.put(Context.PROVIDER_URL, "remote+http://127.0.0.1:8080");
// put your username and pw!
       jndiProps.put(Context.SECURITY_PRINCIPAL, "admin123");
       jndiProps.put(Context.SECURITY_CREDENTIALS, "password1$");
        InitialContext initialContext = new InitialContext(jndiProps);
        Object result = initialContext.lookup("jndi/mykey");
        System.out.println(result);
    }
}
```

Add the libraries!

They're in JBOSS_HOME/bin/client/jboss-client.jar



Name: jboss-client

EJB library of version 20.0.1.Final
JMS library of version 20.0.1.Final
TransactionJavaee library of version 20.0.1.Final
JsonProcessing library of version 20.0.1.Final
ConnectorArchitecture library of version 20.0.1.Final

/Users/ronchet/Download/wildfly-20.0.1.Final/bin/client/jboss-client.jar

```
Nov 24, 2021 2:17:56 PM org.wildfly.naming.client.Version <clinit>
INFO: WildFly Naming version 1.0.13.Final
Nov 24, 2021 2:17:56 PM org.wildfly.security.Version <clinit>
INFO: ELY00001: WildFly Elytron version 1.12.1.Final
Nov 24, 2021 2:17:56 PM org.xnio.Xnio <clinit>
INFO: XNIO version 3.8.1.Final
Nov 24, 2021 2:17:56 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.8.1.Final
Nov 24, 2021 2:17:56 PM org.jboss.threads.Version <clinit>
INFO: JBoss Threads version 2.3.3.Final
Nov 24, 2021 2:17:56 PM org.jboss.remoting3.EndpointImpl
<clinit>
INFO: JBoss Remoting version 5.0.18.Final
MyJndiValue
```

OUTPUT

Warning

JNDI access to these data on Wildfly is READ ONLY!

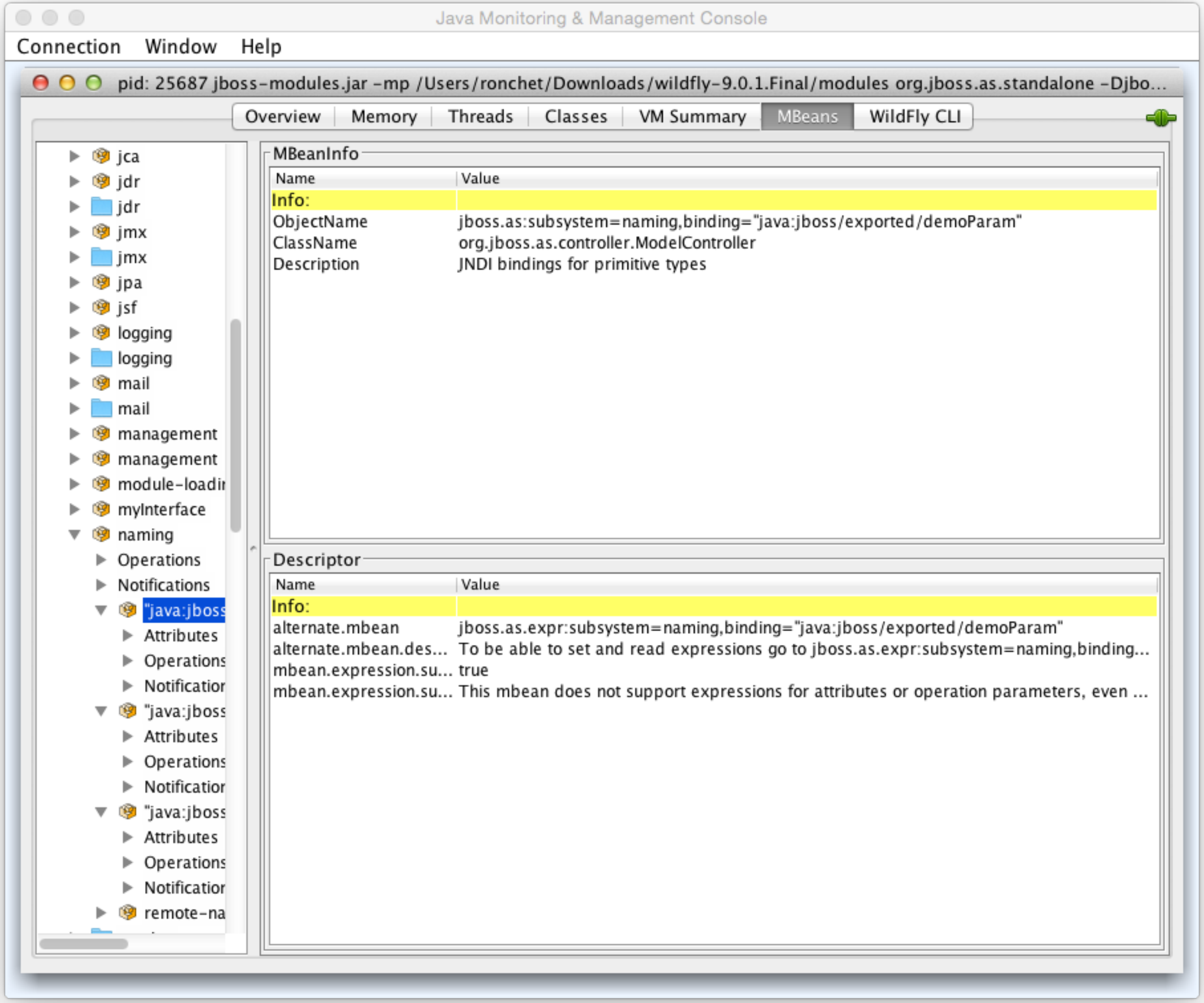
Warning

JNDI access from code is READ ONLY!, but you can write from CLI

```
$ ./jboss-cli.sh
connect
[standalone@localhost:9999 /] /subsystem=naming/binding=
java\:jboss\/exported\/demoParam:add(value=
"Demo configuration value",binding-type=simple)
{ "outcome" => "success"}
[standalone@localhost:9999 /] quit
```

All on one line!

Then reconnect with jconsole to see the result



Name	Value
Info:	
ObjectName	jboss.as:subsystem=naming,binding="java:jboss/exported/demoParam"
ClassName	org.jboss.as.controller.ModelController
Description	JNDI bindings for primitive types

Name	Value
Info:	
alternate.mbean	jboss.as.expr:subsystem=naming,binding="java:jboss/exported/demoParam"
alternate.mbean.des...	To be able to set and read expressions go to jboss.as.expr:subsystem=naming,binding...
mbean.expression.su...	true
mbean.expression.su...	This mbean does not support expressions for attributes or operation parameters, even ...

Useful references

- http://docs.wildfly.org/20/Getting_Started_Guide.html
- http://docs.oracle.com/javase/jndi/tutorial/trailmap.html
- https://docs.jboss.org/author/display/WFLY/Naming%20Subsystem%20Configuration.html