

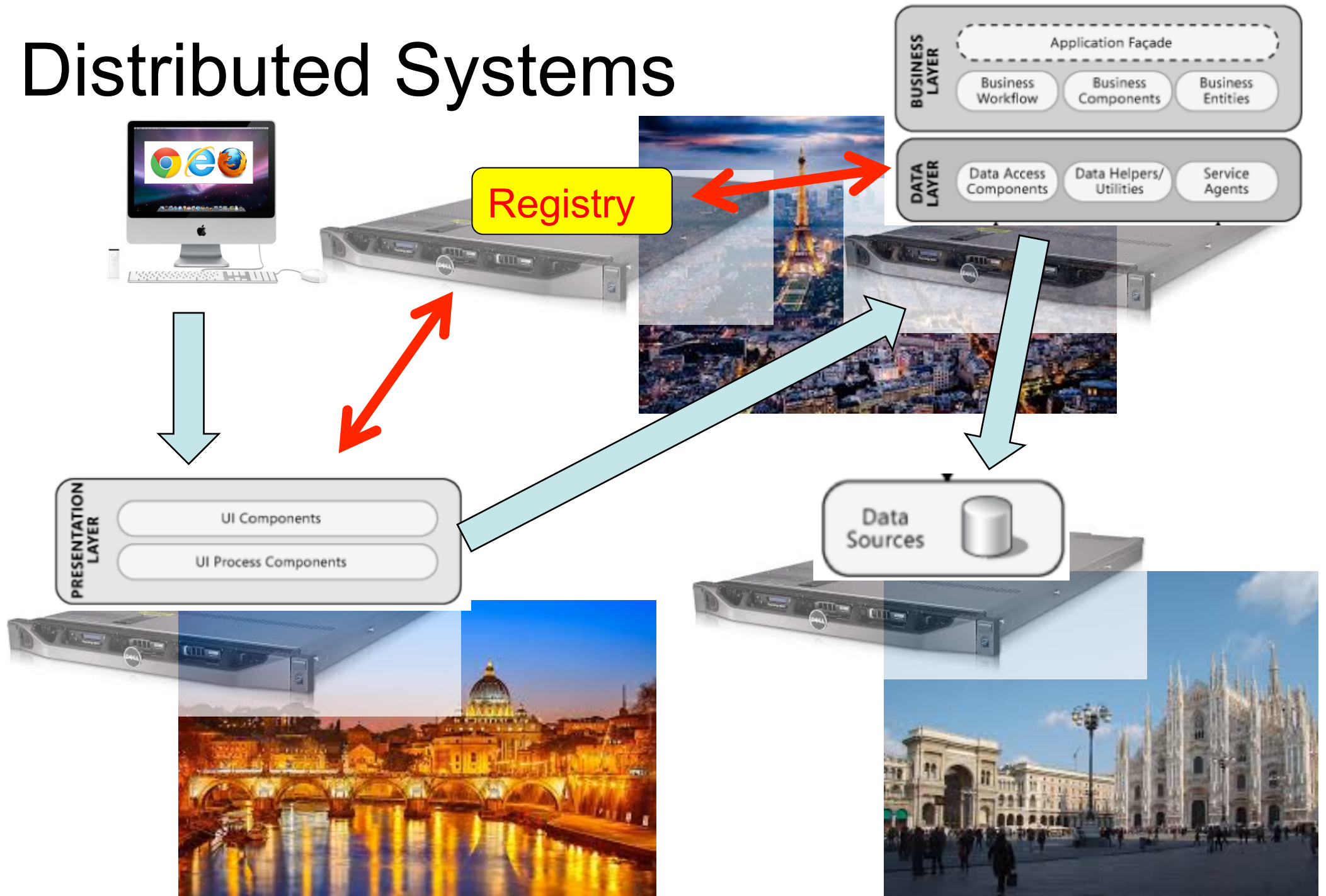
JNDI

Java Naming and Directory Interface

See also:

<http://docs.oracle.com/javase/jndi/tutorial/>

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

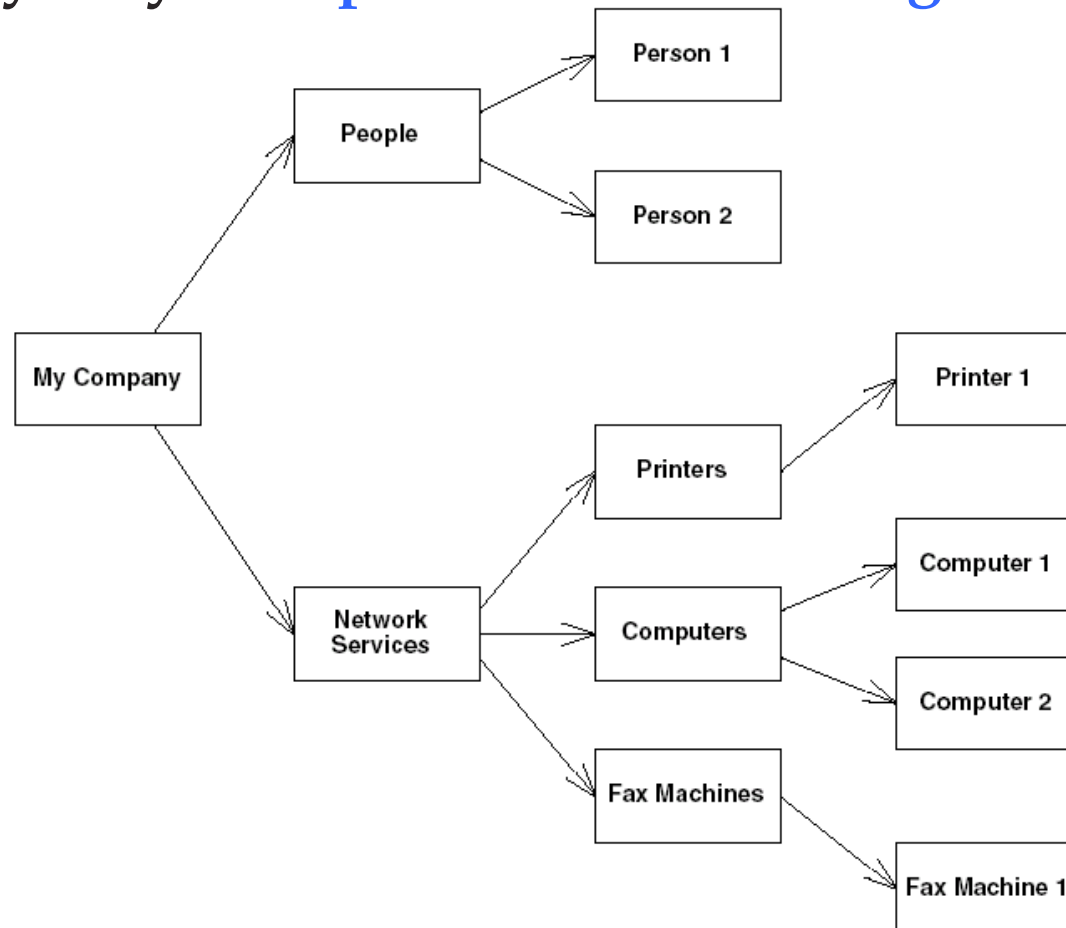
A ***directory object*** differs from a generic object because you can store *attributes* with directory objects. For example, you can use a directory object to represent a user in your company. You can store information about that user, like the user's password, as attributes in the directory object.

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

Netscape Directory Server

Microsoft 's Active Directory

Lotus Notes (IBM)

NIS (Network Information System) by Sun

NDS (Network Directory Service) by Novell

LDAP (Lightweight Directory Access Protocol)

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

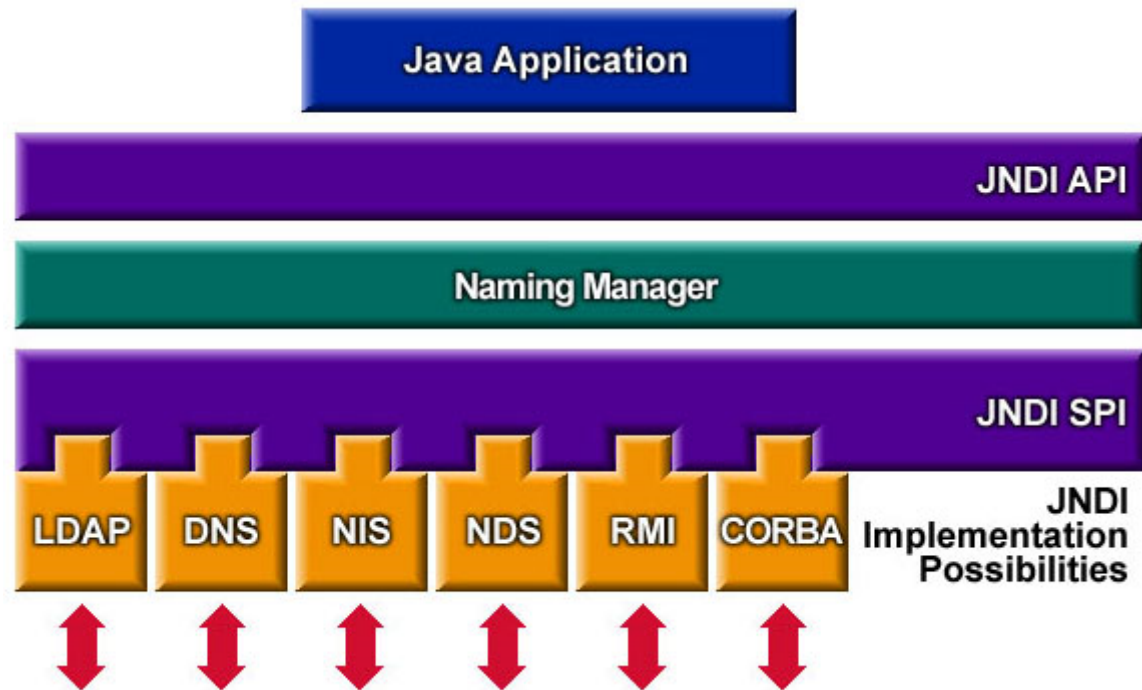
Applications can store factory objects and configuration variables in a global naming tree using the JNDI API.

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



The JNDI homepage

<http://java.sun.com/products/jndi>

has a list of service providers.

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 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

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

Contexts and Subcontexts

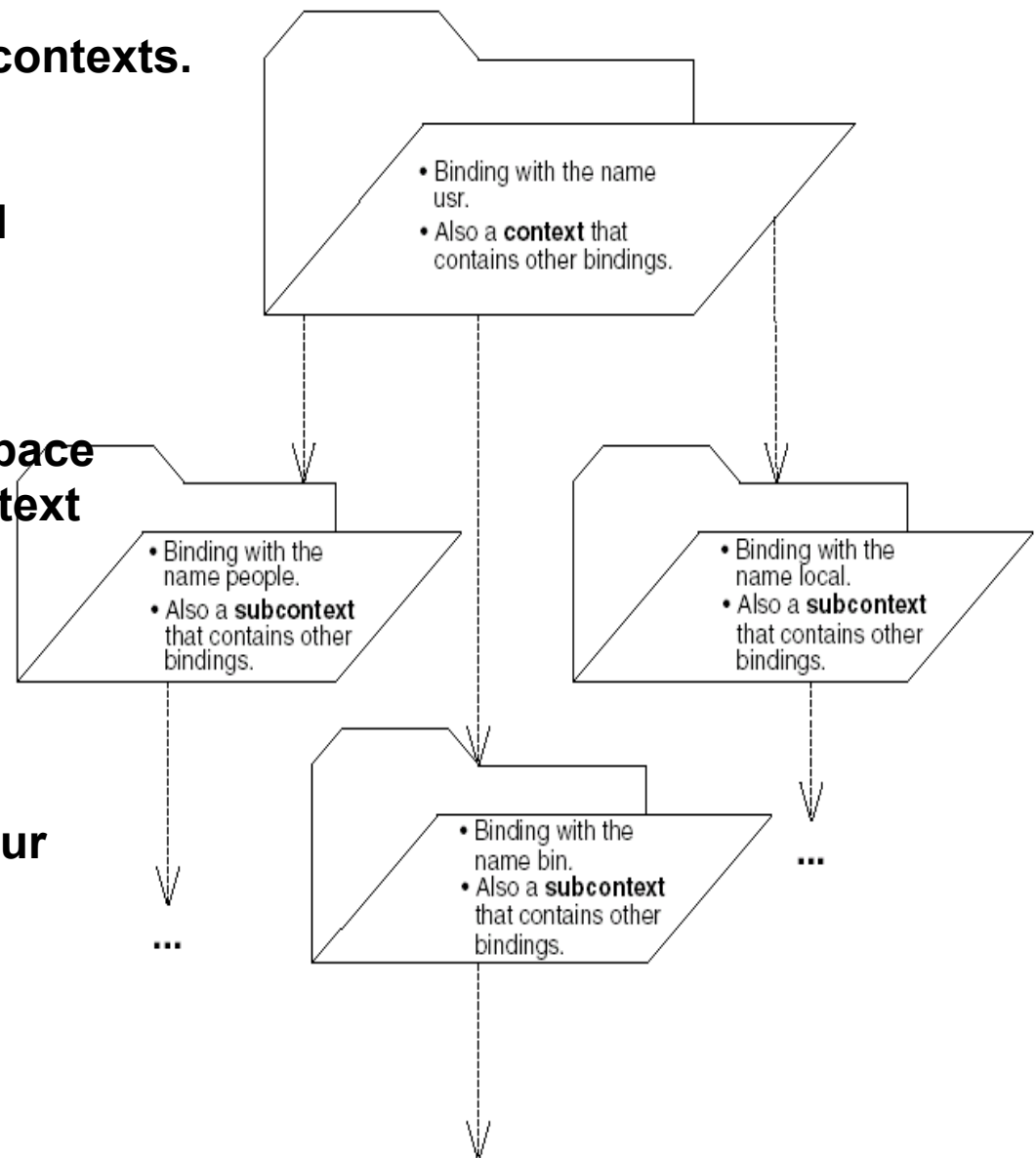
A **naming system** is a connected set of contexts.

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.



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

www.jboss.org

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

Acquiring an initial context

java.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.

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 rmiregistry

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

```
package jndiaccessstormiregistry;
```

```
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 rmiregistry

```
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

Using JNDI to access rmiregistry

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

rmiregistry is FLAT – no subcontexts!

An interesting additional reading about rmiregistry:

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

JNDI Examples

Accessing LDAP

A JNDI-LDAP example

```
package jndiaccessstoldap;
import javax.naming.Context;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.DirContext;
import javax.naming.directory.Attributes;
import javax.naming.NamingException;
import java.util.Hashtable;
public class Getattr {
    public static void main(String[] args) {
        // Identify service provider to use
        Hashtable env = new Hashtable(11);
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.LdapCtxFactory");
        //env.put(Context.PROVIDER_URL, "ldap://ldap.unitn.it:389/o=JNDITutorial");
        env.put(Context.PROVIDER_URL, "ldap://ldap.unitn.it:389/o=personale");
```

```
    try {
        // Create the initial directory context
        DirContext ctx = new InitialDirContext(env);

        // Ask for all attributes of the object
        Attributes attrs = ctx.getAttributes("cn=Ronchetti Marco");

        // Find the surname ("sn") and print it
        System.out.println("sn: " + attrs.get("sn").get());

        // Close the context when we're done
        ctx.close();
    } catch (NamingException e) {
        System.err.println("Problem getting attribute: " + e);
    }
}}
```

A JNDI example on an open LDAP

```
public class Demo {
```

```
    public static void main(String[] args) throws Exception {
        // Identify service provider to use
        Properties env = new Properties();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://ldap.virginia.edu");
        // Create the initial directory context
        DirContext ctx = new InitialDirContext(env);
        list(ctx, "o=University of Virginia,c=US");
        DirContext ctx1 = (DirContext) ctx.lookup("o=University of Virginia,c=US");
        list(ctx1, "ou=Arts & Sciences Graduate");
        DirContext ctx2 = (DirContext) ctx1.lookup("ou=Arts & Sciences Graduate");
        list(ctx2, "ou=casg");
        DirContext ctx3 = (DirContext) ctx2.lookup("ou=casg");
        Attributes attrs = ctx3.getAttributes("cn=Amy Marion Coddington (amc4gc)");
        NamingEnumeration<? extends Attribute> ne = attrs.getAll();
        while (ne.hasMore()) {
            System.out.println(ne.next());
        }
        ctx.close();
    }
}
```

```
static void list(DirContext ctx, String listKey) throws Exception {
    NamingEnumeration<NameClassPair> cp = ctx.list(listKey);
    while (cp.hasMore()) {
        System.out.println(cp.next());
    }
    System.out.println("=====");
}
```

<http://its.virginia.edu/network/publicldap.html>

XML

A quick reminder

Well formed documents

All XML documents must be well-formed

XML documents need not be valid, but all XML documents must be well-formed.

(HTML documents are not required to be well-formed)

There are several requirements for an XML document to be well-formed.

Well formed documents

Caution: XML is case sensitive

Start and end tags are required

To be well-formed, all elements that can contain **character data** must have both **start and end tags**.

(Empty elements have a different requirement: see later.)

For purposes of this explanation, let's just say that the *content* that we discussed earlier comprises character data.

Elements must nest properly

If one element contains another element, the entire second element must be defined inside the start and end tags of the first element.

Well formed documents

Dealing with empty elements

We can deal with empty elements by writing them in either of the following two ways:

```
<book></book>  
<book/>
```

You will recognize the first format as simply writing a start tag followed immediately by an end tag with nothing in between.

The second format is preferable

Empty element can contain attributes

Note that an empty element can contain one or more attributes inside the start tag:

```
<book author="eckel" price="$39.95" />
```

Well formed documents

No markup characters are allowed

For a document to be well-formed, it must not have some characters (**entities**) in the text data: < > “ ‘ &.

If you need for your text to include the < character you can represent it using < or < or < instead.

All attribute values must be in quotes (apostrophes or double quotes).

You can surround the value with apostrophes (single quotes) if the attribute value contains a double quote. An attribute value that is surrounded by double quotes can contain apostrophes.

XML: additional elements

An XML document must have a root tag.

An XML document can contain:

Processing Instructions (PI):

`<? ... ?>`

Comments

`<!-- ... -->`

When a XML document is analyzed, character data within comments or PIs are ignored.

The content of comments is ignored, the content of PIs is passed on to applications.

XML: CDATA sections

Note: the element content that are going to be parsed are called **PCDATA**

An XML document can contain sections used to escape character strings that may contain elements that you do not want to be examined by your XML engine, e.g. special chars (<) or tags:

CDATA sections

<![CDATA[...]]>

When a XML document is analyzed, character data within a CDATA section are not parsed, by they remain as part of the element content.

<java>

<![CDATA[

if (arr[indexArr[4]]>3) System.out.println("<HTML>");

]]>

</java>

Avoid having]]> in your CDATA section!

JBOSS/Wildfly

Java Naming and Directory Interface

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 9.01.Final

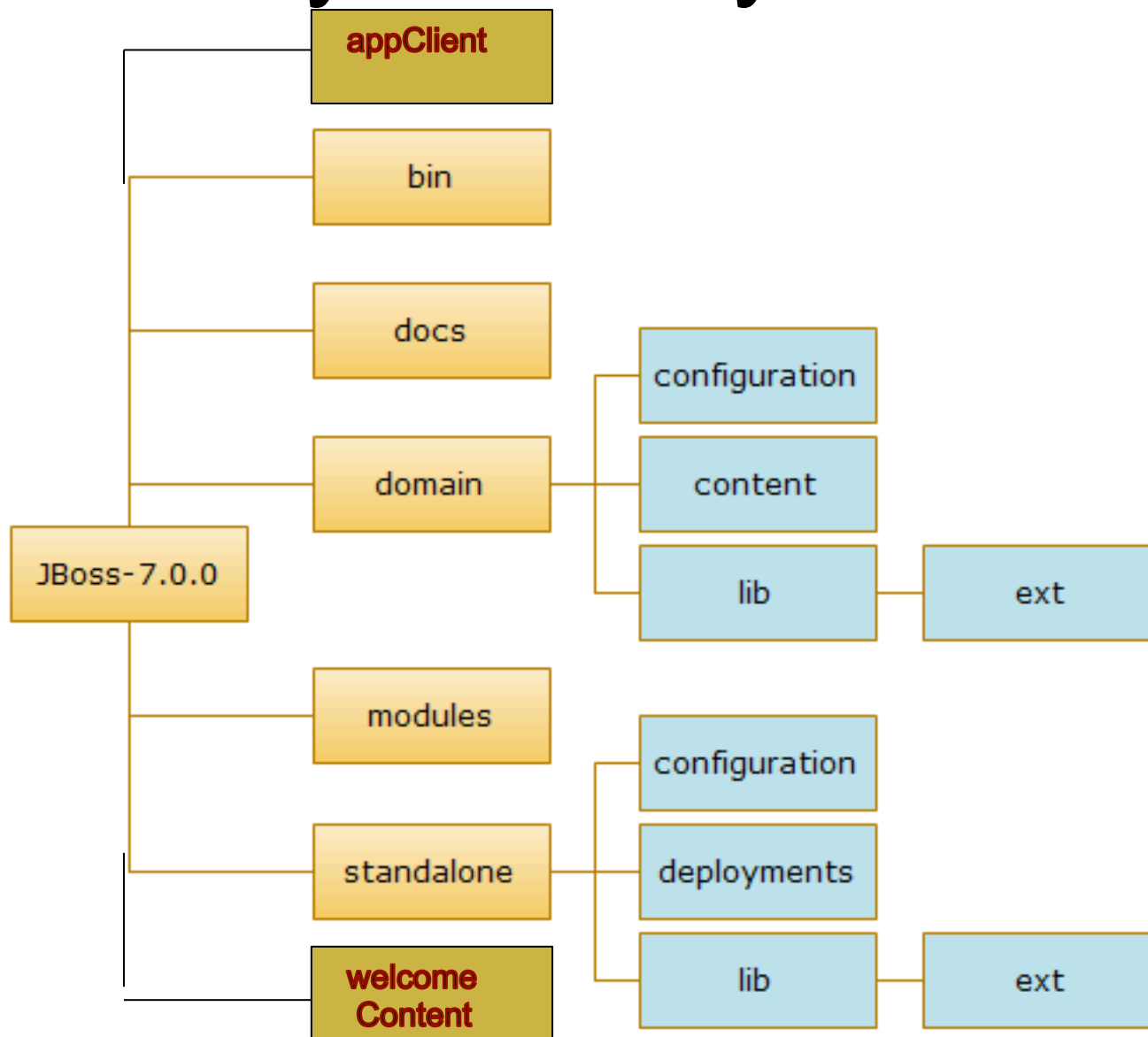
.

GETTING STARTED GUIDE:

<https://docs.jboss.org/author/display/WFLY9/Documentation>

- unzip it (where you like: that will be your JBOSS_HOME)

Wildfly directory structure



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-9.0.1.Final

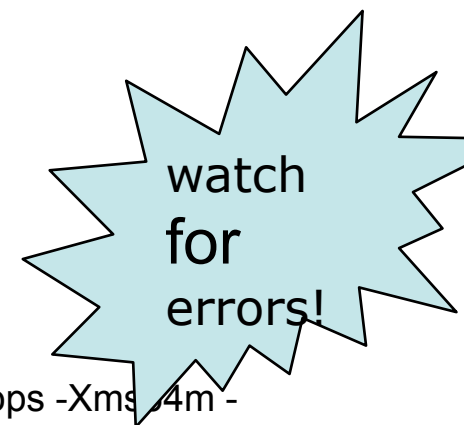
- JAVA: java

- JAVA_OPTS: -server -XX:+UseCompressedOops -server -XX:+UseCompressedOops -Xms4m -Xmx512m -XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true

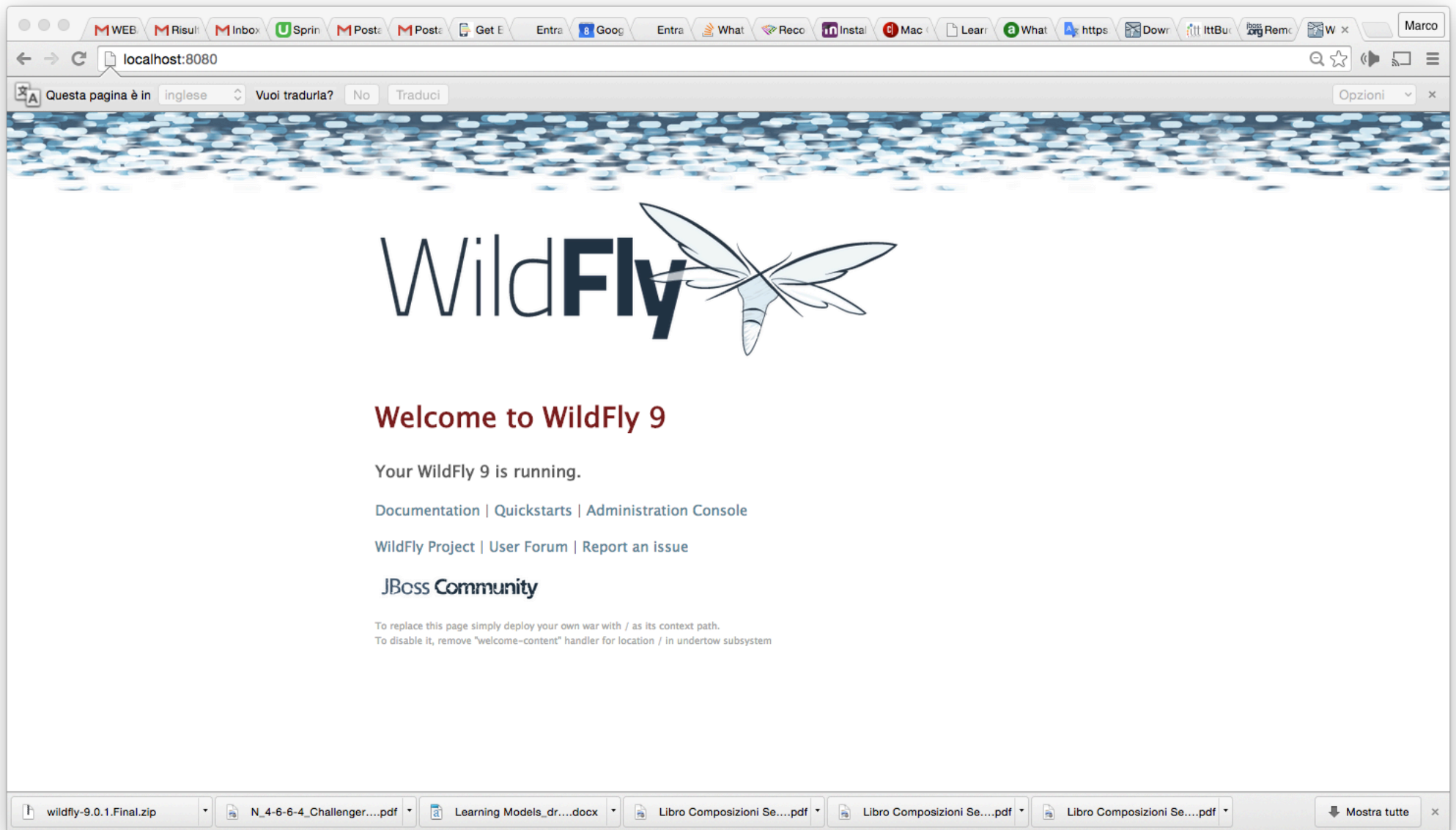
- =====

- Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=256m; support was removed in 8.0
- 22:43:16,319 INFO [org.jboss.modules] (main) JBoss Modules version 1.4.3.Final
- 22:43:16,565 INFO [org.jboss.msc] (main) JBoss MSC version 1.2.6.Final
- 22:43:16,662 INFO [org.jboss.as] (MSC service thread 1-6) WFLYSRV0049: WildFly Full 9.0.1.Final (WildFly Core 1.0.1.Final) starting...
- (...)

22:43:19,264 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: WildFly Full 9.0.1.Final (WildFly Core 1.0.1.Final) started in 3303ms - Started 205 of 382 services (211 services are lazy, passive or on-demand)



http://localhost:8080



Wildfly: Creating admin users

```
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.

Realm (ManagementRealm) : - *leave this blank-*

Username : **admin**

Password : **pw**

Re-enter Password : **pw**

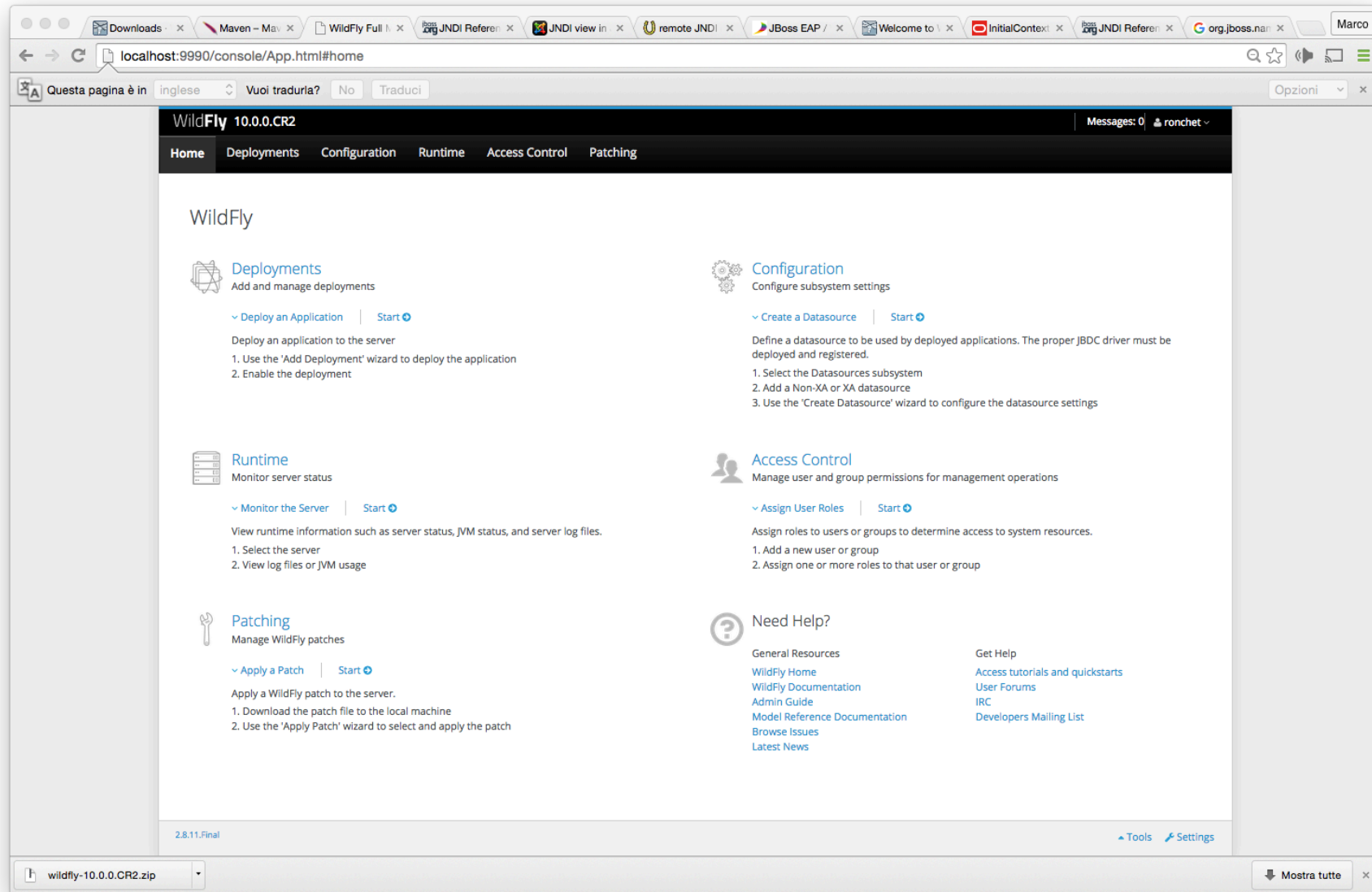
About to add user 'admin' for realm 'ManagementRealm'

Is this correct yes/no? **yes**

Added user 'admin' to file '/Users/ronchet/Downloads/jboss-as-7.1.1.Final/standalone/configuration/mgmt-users.properties'

Added user 'admin' to file '/Users/ronchet/Downloads/jboss-as-7.1.1.Final/domain/configuration/mgmt-users.properties'

http://localhost:9990



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.



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 users

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): **b**

Enter the details of the new user to add.

Realm (ApplicationRealm) : **- leave this blank-**

Username : **user**

Password : **pw**

Re-enter Password : **pw**

What roles do you want this user to belong to? (Please enter a comma separated list, or leave blank for none) : **- leave this blank-**

About to add user 'user' for realm 'ApplicationRealm'

Is this correct yes/no? **yes**

Added user 'user' to file '/Users/ronchet/Downloads/jboss-as-7.1.1.Final/standalone/configuration/application-users.properties'

Added user 'user' to file '/Users/ronchet/Downloads/jboss-as-7.1.1.Final/domain/configuration/application-users.properties'

Adding JNDI bindings

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

```
<subsystem xmlns="urn:jboss:domain:naming:1.1"/>
```

2) replace it with the following section

```
<subsystem xmlns="urn:jboss:domain:naming:1.1">
```

```
  <bindings>
```

```
    <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

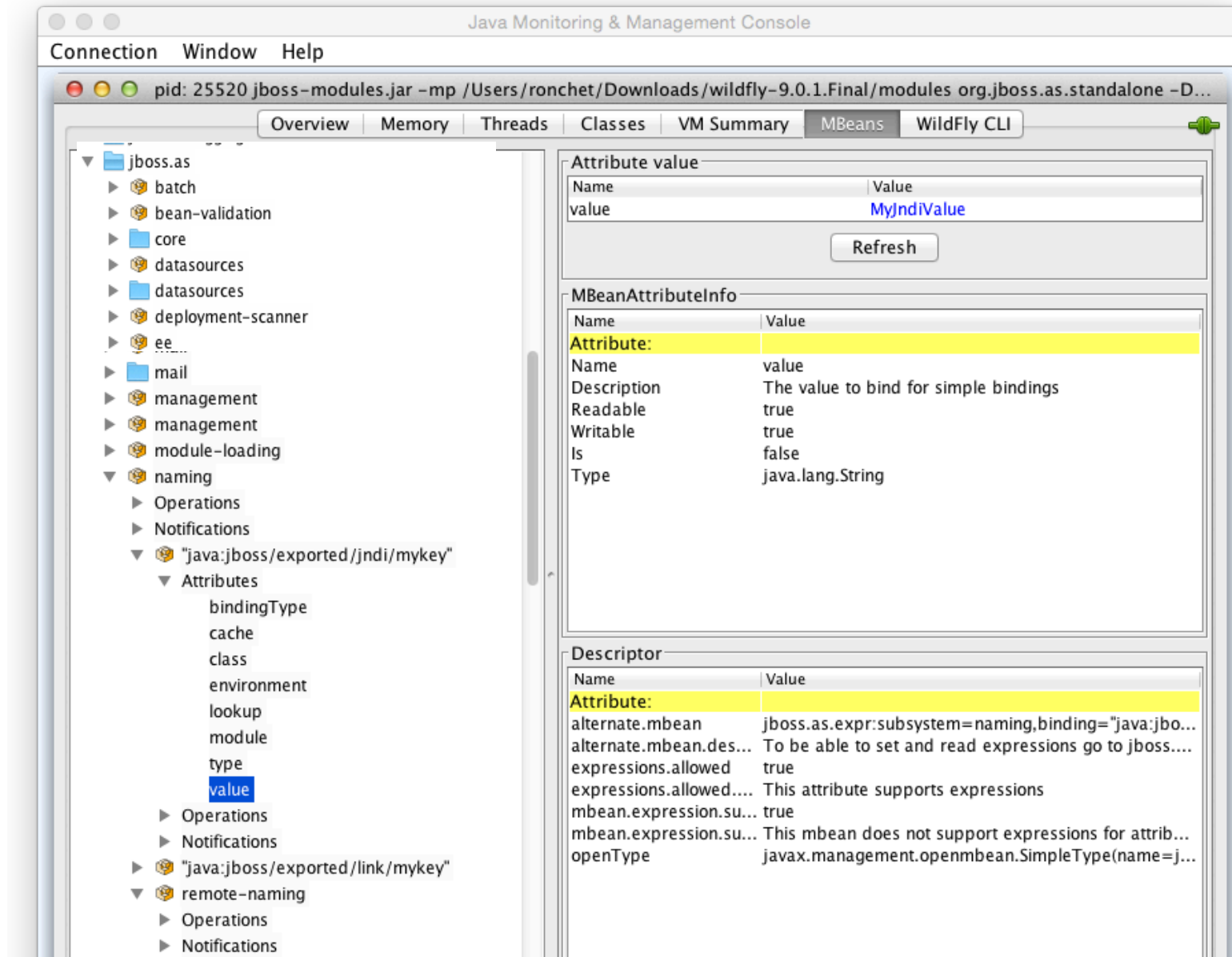
NOTE: the space visible on the client is the one following java:jboss/exported/

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

On client: jndi/mykey

1. start javaconsole (It's in the bin directory)
2. connect with your wildfly running instance
3. inspect the jndi bindings

Inspecting your server



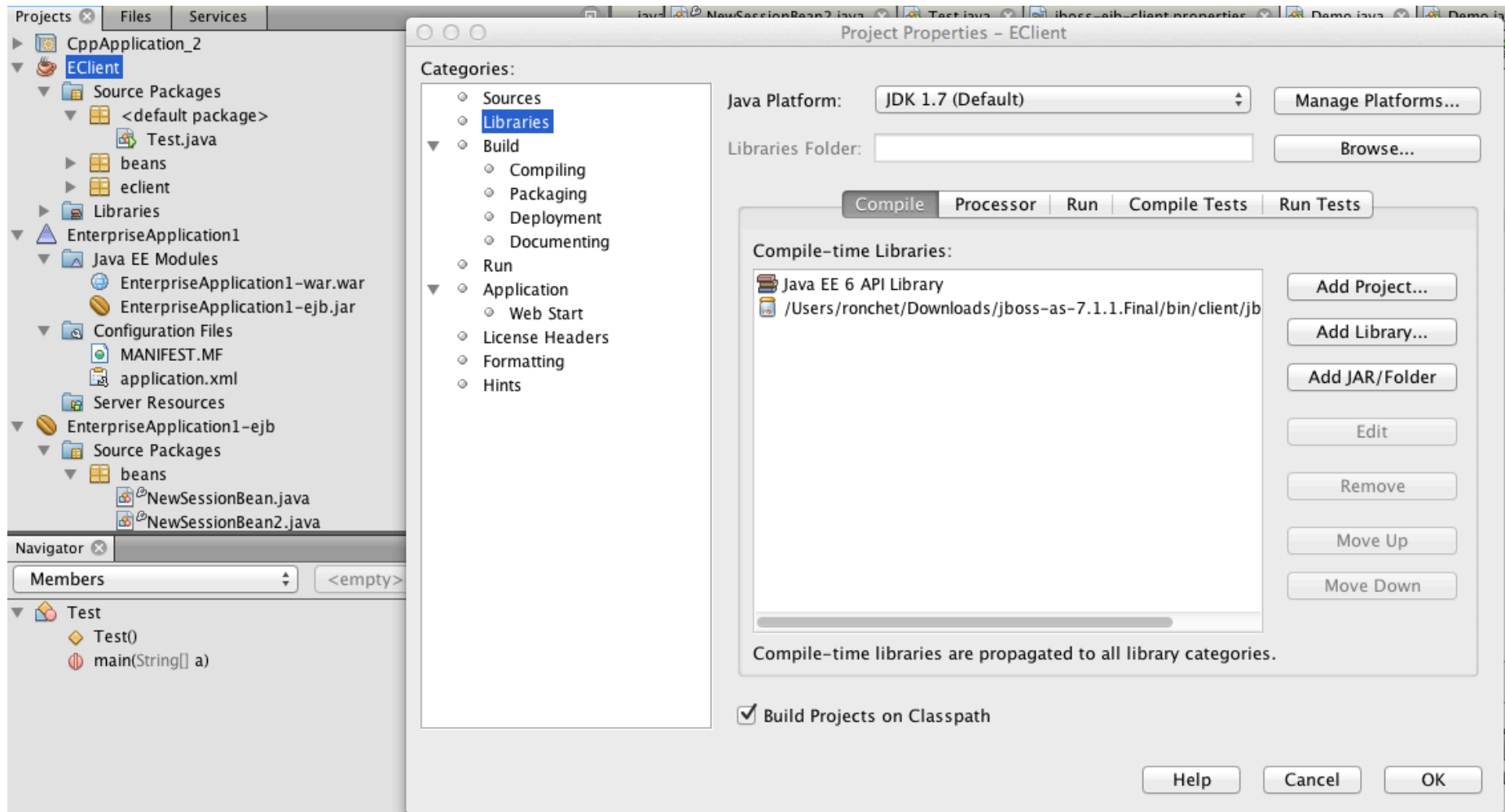
Accessing via code

```
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.jboss.naming.remote.client.InitialContextFactory");  
        jndiProps.put(Context.PROVIDER_URL, "http-remoting://127.0.0.1:8080");  
        // put your username!  
        jndiProps.put(Context.SECURITY_PRINCIPAL, "admin");  
        // put your password!  
        jndiProps.put(Context.SECURITY_CREDENTIALS, "pippo123!");  
        InitialContext initialContext = new InitialContext(jndiProps);  
        Object result = initialContext.lookup("jndi/mykey");  
        System.out.println(result);  
    }  
}
```

see <https://docs.jboss.org/author/display/WFLY8/Remote+EJB+invocations+via+JNDI+-+EJB+client+API+or+remote-naming+project>

Add the libraries!

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



run:

ott 08, 2015 10:58:08 PM org.xnio.Xnio <clinit>

INFO: XNIO version 3.3.1.Final

OUTPUT

ott 08, 2015 10:58:08 PM org.xnio.nio.NioXnio <clinit>

INFO: XNIO NIO Implementation Version 3.3.1.Final

ott 08, 2015 10:58:08 PM org.jboss.remoting3.EndpointImpl <clinit>

INFO: JBoss Remoting version 4.0.9.Final

MyJndiValue

BUILD SUCCESSFUL (total time: 0 seconds)

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

Java Monitoring & Management Console

Connection Window Help

pid: 25687 jboss-modules.jar -mp /Users/ronchet/Downloads/wildfly-9.0.1.Final/modules org.jboss.as.standalone -Djbo...

OverviewMemoryThreadsClassesVM SummaryMBeansWildFly CLI

jcajdrjdrjmxjmxjpajsfloggingloggingmailmailmanagementmanagementmodule-loadimylInterfacenamingOperationsNotifications"java:jboss"AttributesOperationsNotification"java:jboss"AttributesOperationsNotification"java:jboss"AttributesOperationsNotificationremote-na

MBeanInfo

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

Descriptor

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://www.mastertheboss.com/jboss-server/jboss-as-7/jboss-as-7-introduction>
- <https://docs.jboss.org/author/display/AS7/Getting+Started+Guide>
- <https://docs.jboss.org/author/display/AS71/JNDI+Reference>
- <https://docs.jboss.org/author/display/AS71/EJB+invocations+from+a+remote+client+using+JNDI>
- <https://docs.jboss.org/author/display/AS71/Developer+Guide#DeveloperGuide-UpdateapplicationJNDInamespacenames>
- <http://docs.oracle.com/javase/jndi/tutorial/trailmap.html>
- <https://docs.jboss.org/author/display/AS71/CLI+Recipes>