Java Naming and Directory Interface

A JNDI távoli objektumok elérésre szolgál, függetlenül attól, hogy azok Java nyelven íródtak vagy sem.

Context és InitialContext

A Context interfész létfontosságú szerepet játszik a JNDI-ben. Egy context egy bindings(kötés)-sorozatot képez egy konkrét névszolgáltatáson belül. Egy Context objektum metódusokat szolgáltat az objektum binding nevének, az objektum elérését biztosítja amikor hivatkozás történik annak nevére, kilistázza a használt binding-eket és biztosítja a bindin-gek esetleges átnevezését.

A JNDI a context-re nézve relatívan hajtja végre az összes névszolgáltatást. Induláskor a JNDI specifikációk egy InitialContext osztályt értelmeznek. Ezen osztály instanciálva van azon tulajdonságokkal kezdődően, amelyek értelmezik a névszolgáltatás típusát , és ahol szükséges a kezdeti kontextushoz való kapcsolódás pillanatában, a felhasználóinév és jelszó, biztonsági elemeket is használ.

Az InitialContext értelmezési tulajdonságai

A kezdeti kontextus létrehozásának pillanatában, ennek konstruktora egy sor inicializálási információt kell átvegyen. Ezen információk szolgáltatása TulajdonsagNev=TulajdonsagErtek formában történik. A TulajdonsagNev két legfontosabb értéke:

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

Például:

```
java.naming.provider.url=rmi://localhost:1099
java.naming.factory.initial=com.sun.jndi.rmi.regi
stry.RegistryContextFactory
```

A factory osztály különböző szolgáltatásokhoz leggyakrabban használt TulajdonsagErtek-eket tartalmazza az alábbi táblázat:Ezen tulajdonságokon kívül vannak mások is, de azokat ritkán használjuk.

Service	Factory
Filesystem	com.sun.jndi.fscontext.FSContextFactory sau com.sun.Jndi.fscontext.RefFSContextFactory
RMI registry	com.sun.jndi.rmi.registry.RegistryContextFactory
COS	com.sun.jndi.cosnaming.CNCtxFactory
DNS	com.sun.jndi.dns.DnsContextFactory
LDAPv3	com.sun.jndi.ldap.LdapCtxFactory
NIS	com.sun.jndi.nis.NISCtxFactory
NDS	com.novell.naming.service.nds.NdsInitialContextFactory

Ezen tulajdonságokon kívül vannak mások is, de azokat ritkán használjuk.

Hogyan adjuk át a különböző tulajdonságokat a JNDI-nek?

A kezdeti kontextus felépítéséhez szükséges tulajdonságok négy módon adhatók át:

1. egy Hashtable objektum létrehozásával, amely tartalmazza a tulajdonságokat, és amelyet bemenő paraméterként adunk át az InitialContext-nek
Például:

```
Hashtable ht = new Hashtable();
ht.put("java.naming.factory.initial","com.sun.jnd
i.rmi.registry.RegistryContextFactory");
ht.put("java.naming.provider.url",
"rmi://localhost:1099");
Context ctx = new InitialContext(ht);
```

vagy

```
Hashtable ht = new Hashtable();
ht.put(Context.
INITIAL_CONTEXT_FACTORY,"com.sun.jndi.rmi.registr
y.RegistryContextFactory");
ht.put(Context.PROVIDER_URL,
"rmi://localhost:1099");
Context ctx = new InitialContext(ht);
```

2. ezen tulajdonságok hozzáadásával a rendszertulajdonságokhoz.Ez kétféleképpen lehetséges:
Parancssorban-System.setProperty metódus használatával
Példa:

```
System.setProperty("java.naming.factory.initial",
"com.sun.jndi.rmi.registry.RegistryContextFactory
");
System.setProperty("java.naming.provider.url","rm
i://localhost:1099");
```

vagy

```
System.setProperty(Context.
INITIAL_CONTEXT_FACTORY,"com.sun.jndi.rmi.registr
y.RegistryContextFactory");
System.setProperty(Context.PROVIDER_URL,"rmi://lo
calhost:1099");
```

És csak ez(ek) után:

```
Context ctx = new InitialContext();
```

3. a paraméterek egy ?APPLET tag általi értelmezésével, abban az esetben, ha a kliens egy JNDI appletet használ
Példa:

```
<applet code = "Prog.class" width = 600 height =
100 >
<param name="java.naming.factory.initial"
value="com.sun.jndi.rmi.registry.RegistryContextF
actory">
<param name="java.naming.provider.url"
value="rmi://localhost:1099">
</applet>
```

4. a tulajdonságok egy lokális erőforrásfileban való specifikálásával
Egy JNDI-t használó alkalmazás futtatásakor, ez a classpath által látható könyvtárakban keresi a jndi.properties nevu filet. Ezen file minden sorát, mint tulajdonságot értelmezi. Tehát a jndi.properties file tartalma lehet például:

```
java.naming.factory.initial=com.sun.jndi.rmi.regi
stry.RegistryContextFactory
Java.naming.provider.url=rmi://localhost:1099
```

Műveletek attributumokkal

Úgy a DirContext, mint az InitialContext az attributumokkal való műveletekhez specifikus metódusokat ajánl. A legfontosabb metódusok:

```
void bind String nev, Object objektum, Attributes
attributumok)
void rebind String nev, Object objektum,
Attributes attributumok)
DirContext createSubcontext(String nev,
Attributes attributumok)
Attributes getAttributes(String nev)
Attributes getAttributes(String nev, String
[]AttributumNevek)
void modifyAttributes(String nev, int muvelet,
Attributes attributumok)
```

A konstans muveletek:

```
ADD_ATTRIBUTE,
REPLACE_ATTRIBUTE,
REMOVE_ATTRIBUTE.
```

JDBC használata JNDI segítségével

A JDBC 2,0-s verziójától kezdődően egy sor új dolog jelent meg a javax.sql csomagban. Ezek közül említésre méltó: JDBC-ben való JNDI használat, kapcsolatokkal és osztott tranzakciókkal kapcsolatos műveletek. Egy driver, amelyik támogatja a JDBC 2.0-t, kell tartalmazza legalább a DataSource interfész implementálását. Napjaink drivereinek többsége implementálja a JDBC-nek ezt a bővítését.
Na, és miért fontoz ez számunkra? Mert a DataSource által egyszerübben tudunk kapcsolódni az adatbázishoz, mint a DriverManager használata által. DataSource használata tipikus JNDI:

```
javax.naming.Context cx = new InitialContext();
javax.sql.DataSource ds
=(DataSource)cx.lookup("JNDIAdatBazisNev");
javax.sql.Connection co = ds.getConnection();
```

A kapcsolat létrejötte után minden ugyanúgy zajlik, mint ahogy az Adatbázisok Java-ban címü fejezetben ecseteltem. Természetesen ezután a kapcsolattal kapcsolatos egyéb konfigurációk mind a JNDI feladatkörévé válnak.

Az aktuális törekvés a DriverManagertol való fokozatos megválás és a DataSource előtérbe helyezése. Napjainkig nagyon sok modern adatbáziskezelő: **ORACLE, MySQL, MSSQLServer**,sőt az **Access** is lehetővé tette az adatbázishoz való kapcsolódást JNDI által. Úgyszintén, jópár technológia, mint a Tomcat és EJB implementálták a JDBC-hez szükséges JNDI interfészeket.

Finta Anna Maria