Das Java Naming and Directory Interface (JNDI) ist eine Programmierschnittstelle (API) innerhalb der Programmiersprache Java für Namensdienste und Verzeichnisdienste. Mithilfe dieser Schnittstelle können Daten und Objektreferenzen anhand eines Namens abgelegt und von Nutzern der Schnittstelle abgerufen werden. Die Schnittstelle ist dabei unabhängig von der tatsächlichen Implementierung. JNDI ist ein Service Provider Interface (SPI), das Herstellern erlaubt, eigene Lösungen in dieses Framework einzubinden. In der Praxis wird JNDI vor allem dazu benutzt, im Rahmen von Java Enterprise Anwendungen (Java EE 5) verteilte Objekte in einem Netzwerk zu registrieren und sie für Remote-Aufrufe (RMI) weiteren Java-Programmteilen zur Verfügung zu stellen.

Inhaltsverzeichnis

1. Einleitung

Der Java Naming und Directory Service ermöglicht die Konfiguration von Fabriken, bei denen zur Laufzeit Java-Objekte durch die Angabe eines Pfades in Form einer Zeichenkette angefordert werden können. Die Schnittstelle (JNDI) ist dabei unabhängig von der tatsächlichen Implementierung. JNDI enthält ein Service Provider Interface (SPI), das Herstellern erlaubt, eigene Lösungen (Fabriken) in dieses Framework einzubinden. Eine Liste solcher Fabriken finden Sie bei Sun unter Java Naming and Directory Interface (JNDI) Service Providers.

1.1 Namensdienste

Ein Namensdienst liefert zu einem bestimmten Namen genau ein oder kein Objekt. Der Domain Name Service (DNS) ist ein solcher Dienst: Zu einem Host-Namen (z.B. `www.hft-so.ch`) liefert er die IP-Adresse (`194.121.37.110`). Ein Name dient also dazu, ein beliebiges Objekt genau zu identifizieren.

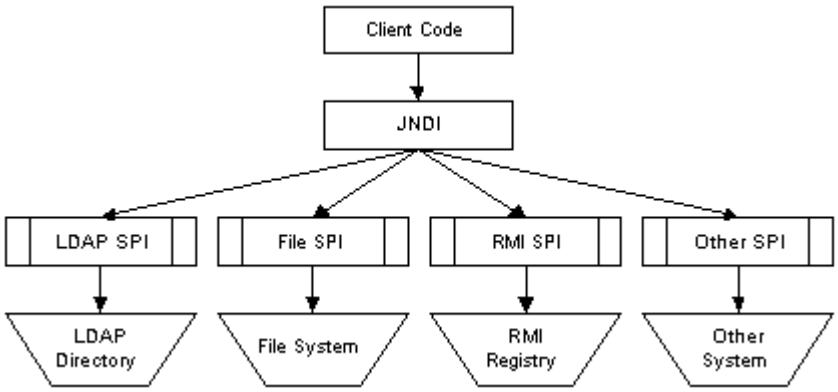
1.2 Verzeichnisdienste

Ein Verzeichnisdienst stellt die Informationen in einer hierarchischen Struktur zur Verfügung. Ein einfaches Beispiel ist das Dateisystem auf Unix: Unter dem Wurzelknoten / steht ein Baum aus Verzeichnissen. Ein weiteres Beispiel ist das Lightweight Directory Access Protocol (LDAP), mit dem typischerweise auf Benutzerdaten in einem Verzeichnis-Server zugegriffen wird. Die Daten sind dabei hierarchisch strukturiert, beispielsweise in Abteilungen und Unterabteilungen. Ein einzelnes Objekt wird dabei durch einen

eindeutigen Namen gekennzeichnet. In einem Dateisystem ist dies der Pfadname (z.B. `/usr/bin`), bei LDAP der so genannte Distinguished Name oder kurz DN, der den Pfad zu einem Objekt eindeutig bezeichnet (z.B. `uid=bau, ou=Mitarbeiter, o=HFT-SO, c=CH`).

1.3 Architektur

JNDI arbeitet über Links zwischen Namen und Ressourcen. Ähnlich wie JDBC fordert JNDI über Interfaces, was die Hersteller zu implementieren haben.



Auf jeden Verzeichnis- und Namensdienst wird über die Service Provider-Schnittstelle zugegriffen. Die Implementation des Verzeichnis- und des Namensdienstes sieht der Client nicht! Die Spezifikation von JNDI und entsprechende Tutorials sind unter Sun zu finden. Es ist ein muss, das JNDI-Tutorial von Sun durchgearbeitet zu haben!

1.4 Praxiseinsatz

In der Praxis wird JNDI vor allem dazu benutzt, im Rahmen von Java Enterprise Anwendungen (J2EE-Anwendungen) verteilte Objekte in einem Netzwerk zu registrieren und sie für Remote-Aufrufe (RMI) weiteren Java-Programmteilen zur Verfügung zu stellen. JNDI erlaubt die Unterstützung praktisch aller Arten von Namens- und Verzeichnisdiensten. Das folgende Beispiel zeigt den Einsatz eines Verzeichnisdienstes (Dateisystem):

```
public class Lookup {
  public static void main(String[] args) {
    String name = "";
    if (args.length > 0)
      name = args[0];
    try {
      // Erzeugen eines Properties Objekts und setzen
      // der Eigenschaften
      Properties pts = new Properties();
      pts.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      pts.put(Context.PROVIDER_URL, "file:///");
      // Erzeugen eines InitialContext mittels pts
      Context ctx = new InitialContext(pts);
      // Look up auf den eingegebenen Namen
      Object obj = ctx.lookup(name);
      System.out.println(obj);
      // Auflisten der Name-Klasse Paare
```

```
      NamingEnumeration list = ctx.list(name);
      while (list.hasMore()) {
        NameClassPair nc = (NameClassPair) list.next();
        System.out.println(nc.getName() + "\t" + nc.getClassName());
      }
    }
    catch (NamingException ex) {
      System.out.println("Exeption: " + ex.toString());
    }
  }
}
```

- Jede JNDI-Abfrage beginnt zunächst mit dem `InitialContext`. Dieser stellt den Einstiegspunkt dar, unter dem Objekte gesucht und abgelegt werden können. Neben dem Klassennamen des Providers wird die URL zum Einstiegspunkt angegeben (anstelle des Properties-Objekt kann auch eine Hashtable oder eine Ressourcendatei eingesetzt werden).

- Der Context beschreibt eine Menge von Bindungen zwischen Namen und Objekten. So liefert der Context `file:///` eine Liste aller Namen der Dateien und Verzeichnisse unter dem Root-Verzeichnis.

- Mit der Methode `list` kann der Inhalt eines Kontextes ausgelesen werden. Der Ausdruck könnte beispielsweise folgendermassen aussehen:

```
com.sun.jndi.fscontext.RefFSContext@19189e1
AUTOEXEC.BAT java.io.File
Bilder javax.naming.Context
boot.ini java.io.File
bootfont.bin java.io.File
CONFIG.SYS java.io.File
Dokumente und Einstellungen javax.naming.Context
I386 javax.naming.Context
IO.SYS java.io.File
MSDOS.SYS java.io.File
Programme javax.naming.Context
RECYCLER javax.naming.Context
Ruedi javax.naming.Context
...
```

2. Tomcat und JNDI

JNDI wird beim Tomcat Server insbesondere für den Zugriff auf Datenbanken und Mail-Sessions eingesetzt. Tomcat 7 stellt seinen Web-Applikationen eine Instanz des JNDI Initial Context zur Verfügung. Die jeweiligen Einträge des `InitialContext` können in der zentralen Konfigurationsdatei `%CATALINA_HOME%/conf/server.xml` oder in der `context.xml` Datei der jeweiligen Web-Applikation vorgenommen werden (die folgenden Ausführungen beziehen sich auf die zweite Variante!). Dabei werden alle konfigurierten Einträge bzw. Ressourcen in den Namensraum `java:comp/env` eingefügt.

Ein typischer Zugriff auf ein JDBC-DataSource-Objekt sieht damit folgendermassen aus (für die Gästebuch Applikation):

```
    try {
      Context initCtx = new InitialContext();
      Context envCtx = (Context) initCtx.lookup("java:comp/env");
      DataSource ds = (DataSource) envCtx.lookup("jdbc/gaestebuchDB");
      if (ds != null) {
        System.out.println("DataSource: " + ds);
        con = ds.getConnection();
      }
    }
    catch (Exception ex)  {
      System.out.println(ex.toString());
    }
```

2.1 context.xml

Um der Web-Applikation JNDI-Ressourcen bereitzustellen, müssen diese im Verzeichnis `WEB-INF` der Applikation in der Datei `context.xml` eingetragen werden. Die Datei `context.xml` sieht für den Tomcat 7.x folgendermassen aus (Datenbankzugriff auf HSQLDB):

```
<Context>
  <Resource name="jdbc/gaestebuchDB"
            auth="Container"
            type="javax.sql.DataSource"
            factory="org.apache.commons.dbcp.BasicDataSourceFactory"
            maxActive="100"
            maxIdle="30"
            maxWait="10000"
            username="sa"
            password=""
            driverClassName="org.hsqldb.jdbcDriver"
            url="jdbc:hsqldb:hsql://localhost" />
</Context>
```

Leider ist die Dokumentation für den Tomcat 7.x nicht mehr auf dem Stand der Wirklichkeit und verursacht den Anwendern Bauchschmerzen!

2.2 web.xml

Die Konfiguration des Deployment Descriptors der Web-Applikation muss für den Zugriff auf die Ressource entsprechend konfiguriert werden:

```
  <resource-ref>
    <description>Connection to my DB</description>
    <res-ref-name>jdbc/gaestebuchDB</res-ref-name>
    <res-type>javax.sql.DataSource </res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

  <env-entry>
    <description>
      Begrüssungstext über JNDI
    </description>
```

```
    <env-entry-name>
      begruessen
    </env-entry-name>
    <env-entry-type>
      java.lang.String
    </env-entry-type>
    <env-entry-value>
      Ich begrüsse Sie auf meiner Gästebuch-Website
    </env-entry-value>
  </env-entry>
```

2.3 Holen der Datenbank-Ressource

Das Holen einer JDBC Datenbank-Ressource ist im JNDI Resources HOW-TO der Tomcat Dokumentation beschrieben und sieht in der Praxis folgendermassen aus:

```
public class GaestebuchDB {

  private Connection con;
  private boolean freeCon = true;
  private static GaestebuchDB singleton = null;
/*
  private ResourceBundle rb = ResourceBundle.getBundle("database");
  private String driver = rb.getString("DRIVER");
  private String url = rb.getString("URL");
  private String user = rb.getString("USER");
  private String password = rb.getString("PASSWORD");
*/
  private GaestebuchDB() throws Exception {
/*
    try {
      Class.forName(driver);
      // Verbindung holen
      con = DriverManager.getConnection(url, user, password);
    }
    catch (Exception ex) {
      throw new Exception("Die Datenbank kann nicht geoeffnet werden: "
        + ex.getMessage());
    }
*/
    try {
      Context initCtx = new InitialContext();
      Context envCtx = (Context) initCtx.lookup("java:comp/env");
      DataSource ds = (DataSource) envCtx.lookup("jdbc/gaestebuchDB");
      if (ds != null) {
        System.out.println("DataSource: " + ds);
        con = ds.getConnection();
      }
    }
    catch (Exception ex)  {
      System.out.println(ex.toString());
    }
  }
  ...
}
```

3. Zusammenfassung

Das Java Naming and Directory Interface™ (JNDI) ist ein API, welches von Sun den Entwicklern zur Verfügung gestellt wird und die Möglichkeit bietet, Naming und Directory Service Funktionalitäten in Java Anwendungen zu nutzen. Damit definiert Sun eine abstrakte Schnittstelle zum Zugriff auf verschiedene Naming und Directory Services. Eine Java-Anwendung kann durch diese abstrakte Schnittstelle einheitlich auf verschiedene Naming und Directory Services zugreifen. Dadurch können diese ausgetauscht werden ohne die Java-Anwendung neu zu kompilieren (nur wenn die Namen im neuen Service gleich bleiben).

Literatur

The Apache Software Foundation: Apache Tomcat 7 User Guide. Fultus Corp 2011