In diesem Kapitel:

Java Naming und Directory Interfaces

1.1. Einleitung – Grundsätzliches

Directory Services spielen eine vitale Rolle in Intranets und Internets, weil sie den Zugriff auf unterschiedlichste Informationen und 'Hosts' (in Internet Sinne : irgend etwas, was eine IP Adresse besitzt), Netzwerke, Services und Applikationen erlauben oder ermöglichen. Ein **Verzeichnisdienst** umfasst auf Grund seiner eigentlichen Funktion eine *Einrichtung* zum *Nachschlagen* von *Namen* in *Namensräumen* ('naming facility'), welche auch für Menschen lesbar sind und welche unterschiedliche Entitäten zusammenfassen eventuell unter unterschiedlichen Gesichtspunkten.

Typischerweise umfasst eine Rechnerumgebung unterschiedliche Naming Facilities (Einrichtungen zum Nachschlagen von Namen in Namensräumen), mit deren Hilfe komplexe Verzeichnisdienste aufgebaut werden können.

Das Internet *Domain Name System* (DNS) kann auf einer relativ hohen Ebene als übergeordneter Verzeichnisdienst eingesetzt werden, organisationsübergreifend. Daneben wird man aber vermutlich auch Verzeichnisdienste wie LDAP, NDS oder NIS oder ganz einfach Dateisysteme verwenden (siehe unten für eine Erläuterung dieser Begriffe).

Aus Benutzersicht hat man vielleicht ein Adressierungsschema, welches aus *zusammengesetzten* Namen besteht. URLs sind Beispiele für zusammengesetzte Namen.

Viele Java Applikationensentwickler nutzen Verzechnisdienste und setzen diese in ihren Applikationen ein. Ein Verzeichnisdienst API wäre also durchaus etwas Sinnvolles. Dieses API sollte so definiert sein, dass es unterschiedliche Verzeichnisschemata unterstützt.

Es wäre auch denkbar, dass eine Applikation ihre Objekte in einem Verzeichnisdienst einträgt und eine beliebige Java Applikation Objekte belieben Typus mit Hilfe des Verzeichnisdienstes herunterladen könnte.

Der Benutzer kann auch von Verzeichnisdiensten profitieren, da er Objekte (im weitesten Sinne) besser lokalisieren kann und sie dadurch eher nutzt.

Entwickler von Verzeichnisdiensten könnten von *Service- Providern* profitieren, welche diese Dienste direkt anbieten: das spart Entwicklungszeit und erlaubt eine bessere Portabilität, Wartung und eine höhere Rekonfigurationsfähigkeit. Die Software wird damit leichter an neue Situation anpassbar.

JAVA NAMING UND DIRECTORY INTERFACES

Java Naming and Directory Interface (JNDI) ist ein API, welches Java Applikationen Verzeichnis- und Namensgebungs- Funktionalität anbietet. Das API ist Verzeichnisdienst unabhängig, sollte also in Zusammenarbeit mit andern Diensten funktionieren.

Beispiele:

1. eine Applikation möchte ein Dokument drucken und benötigt ein entsprechendes Printer Objekt.

   Das lässt sich schematisch folgendermassen bewerkstelligen:

   ```
   prt = (Printer) D200.lookup("LaserWriter2");
   prt.print(document);
   ```

   Mit Hilfe von JNDI kann das Printerobjekt gefunden und der Anwendung zur Verfügung gestellt werden.

2. Eine Applikation, welche eine Person in einem Telefonverzeichnis sucht, welches als Verzeichnisdienst implementiert ist, könnte etwa folgendermassen aussehen:

   ```
   String[] attrs = {"bueroNummer", "natelNummer",
                     "faxNummer", "privateNummer"};
   nicolesNummer = directory.getAttributes("cn=Nicole, o=ETH,
                           c=CH", attrs);
   ```

   Falls mehrere Nicole's im ETH Verzeichnis eingetragen sind (vermutlich trifft dies zu), kann man weiter suchen:

   ```
   nicolesNummer = directory.search("o=ETH, c=CH", "(cn=Nicole)",
                                    searchctls);
   ```

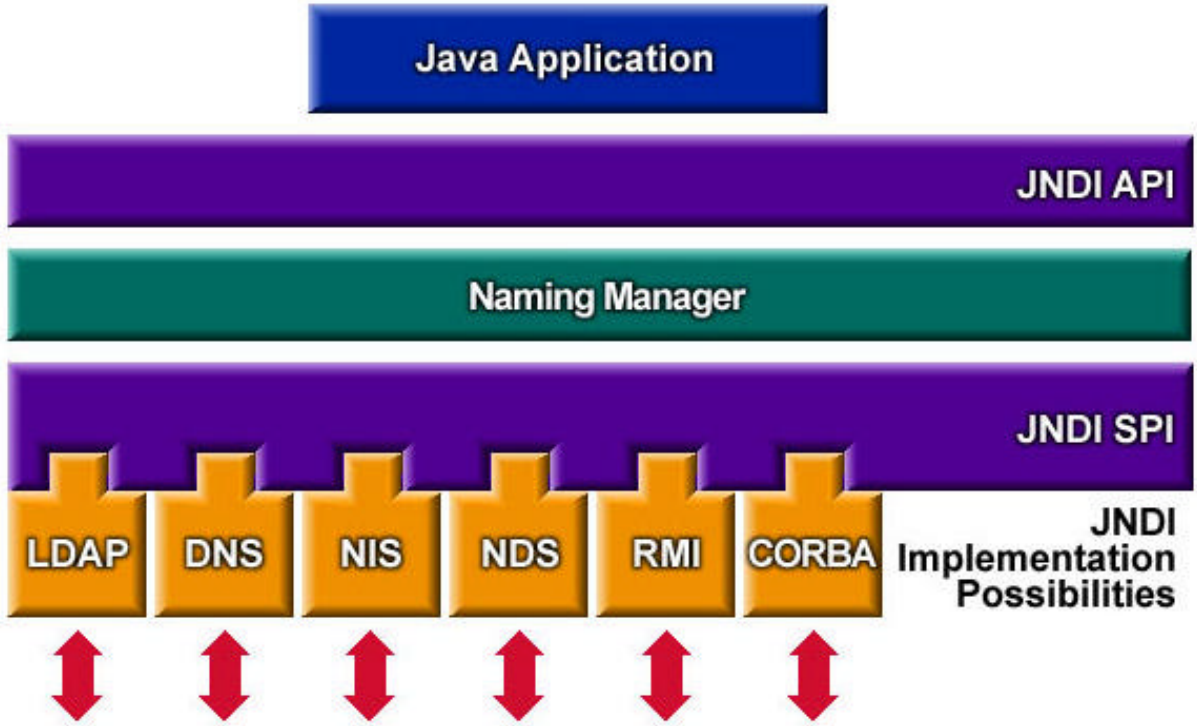
In den folgenden Abschnitten wollen wir die Architektur und die Interfaces von JNDI kennen lernen, sowie andere Verzeichnisdienste mindestens grundsätzlich besprechen.

1.2. JNDI Übersicht

Das Java Naming und Directory InterfaceTM (JNDI) ist ein Application Programming Interface (API), welches Namens- und Verzeichnis- Funktionalitäten in Java zur Verfügung stellen. Es ist weitestgehend unabhängig von spezifischen Verzeichnisdienstimplementationen. Damit hat man die Möglichkeit unterschiedliche Dienste anzuwenden.

1.2.1. Architektur

Die JNDI Architektur besteht aus einem *API* und einem *Service Provider Interface (SPI)*. Java Applikationen verwenden das JNDI API, um auf unterschiedliche Namens- und Verzeichnisdienste zugreifen zu können. Das SPI ermöglicht es einer Anwendung unterschiedliche Namens- und Verzeichnis**dienste** einzusetzen.



Java stellt bereits diverse dieser SPIs zur Verfügung, beispielsweise:

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service
- Java Remote Method Invocation (RMI) Registry

Andere Provider Packages können entweder von Sun oder den entsprechenden Anbietern herunter geladen werden.

JNDI selbst besteht aus den Packages:

javax.naming
javax.naming.directory
javax.naming.event
javax.naming.ldap
javax.naming.spi

1.2.2. Naming Beispiel - Lookup

```
package lookupbsp;
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import javax.naming.*;
import java.io.File;
import java.util.Hashtable;
/**
  * Grundfunktion : Lookup einer Adresse, eines Namens
  *
  * usage: java Lookup
  */
public class Lookup {
  public static void main(String[] args) {
    // Achtung : in URL würde man file://... erwarten
    //           hier muss man file:/ verwenden, also nur ein Slash
    String[] str_Verzeichnis = {"file:/", "file:/Temp"};
    String[] str_Datei = {"index.html", "indexm.html"} ;
    String str_Error = "";
    // die Hashtabelle dient als Basis für den Context
    Hashtable env = new Hashtable(11);
    // fscontext bezieht sich auf das Filesystem (f s )
    env.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.jndi.fscontext.RefFSContextFactory");
    for (int i=0; i< str_Verzeichnis.length; i++) {
      for (int j=0; j< str_Datei.length; j++) {
        System.out.println("Context (Verzeichnis) : "+str_Verzeichnis[i]+
                           "; \nDatei : "+str_Datei[j]);
      //Angabe des Verzeichnisses
      //************************
      env.put(Context.PROVIDER_URL, str_Verzeichnis[i]);
      try {
        // kreieren des initial context mit der RefFSContextFactory
        str_Error = "Context Fehler : "+str_Verzeichnis[i];
        Context ctx = new InitialContext(env);
        // lookup und casting
        //******************
        // das Root war ein Verzeichnis; jetzt geben wir eine Datei an
        str_Error = "Lookup Fehler : "+str_Datei[j]+
                    " nicht gefunden im FSContext "+str_Verzeichnis[i];
        File f = (File) ctx.lookup(str_Datei[j]);
        System.out.println(f);
        // Context schliessen (eigentlich erst am Schluss)
        ctx.close();
      } catch (NamingException e) {
        System.err.println(str_Error);
        System.err.println("NamingException : "+e);
      }
      System.out.println("_______");
      }
    }
  }
}
```

Ausgabe:

```
lookupbsp.Lookup

Context (Verzeichnis) : file:/;
Datei : index.html
_______
Context (Verzeichnis) : file:/;
Datei : indexm.html
Context Fehler : file:/
NamingException : javax.naming.NoInitialContextException: Cannot
instantiate class: com.sun.jndi.fscontext.RefFSContextFactory [Root
exception is java.lang.ClassNotFoundException:
com.sun.jndi.fscontext.RefFSContextFactory]
Context Fehler : file:/
NamingException : javax.naming.NoInitialContextException: Cannot
instantiate class: com.sun.jndi.fscontext.RefFSContextFactory [Root
exception is java.lang.ClassNotFoundException:
com.sun.jndi.fscontext.RefFSContextFactory]_______
Context (Verzeichnis) : file:/Temp;
Datei : index.html
_______
Context (Verzeichnis) : file:/Temp;
Datei : indexm.html

Context Fehler : file:/Temp
NamingException : javax.naming.NoInitialContextException: Cannot
instantiate class: com.sun.jndi.fscontext.RefFSContextFactory [Root
exception is java.lang.ClassNotFoundException:
com.sun.jndi.fscontext.RefFSContextFactory]
Context Fehler : file:/Temp
NamingException : javax.naming.NoInitialContextException: Cannot
instantiate class: com.sun.jndi.fscontext.RefFSContextFactory [Root
exception is java.lang.ClassNotFoundException:
com.sun.jndi.fscontext.RefFSContextFactory]_______
```

1.2.3. LDAP Directory Beispiel - Lesen der Attribute

```
package LesenDerAttribute;

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;

/**
 * Lesen von Attributen eines benannten Objektes.
 *
 * usage: java GetAttr
 */
class GetAttr {
  public static void main(String[] args) {
        // Provider
        Hashtable env = new Hashtable(11);
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://ztnw293:389/o=Attribute");

        try {
          // initialisieren des Contextes
          DirContext ctx = new InitialDirContext(env);
          // Attribute des Objektes abfragen
          Attributes attrs = ctx.getAttributes("cn=Josef Joller,
ou=Personal");
          // suche ("sn") und Ausgabe
          System.out.println("sn: " + attrs.get("sn").get());
          // und tschüss ...
          ctx.close();
        } catch (NamingException e) {
            System.err.println("Problem : Attribute können nicht bestimmt
werden " + e);
        }
  }
}
```

Achtung:
damit das obige Beispiel getestet werden kann, müssen Sie einen LDAP Server kennen oder laufen lassen.
Sie finden eine gratis Implementation mit Links auf eine Datenbank, die das Programm verwendet, alle im Sourcecode in C, auf dem Website http://www.OpenLDAP.org .
Alternativ kann man einen der öffentlichen LDAP Server verwenden:

```
ldap://ldap.Bigfoot.com
ldap://ldap.four11.com
ldap://ldap.InfoSpace.com
ldap://ldap.yahoo.com
```

1.3. Begriffsklärung

Damit Sie die folgenden Beispiele besser verstehen müssen wir zuerst noch einige Begriffe klären. Die Begriffe sind weitestgehend unabhängig von JNDI und gelten universell.

1.3.1. Namenskonzepte

Jeder Rechner benötigt in irgend einer Form einen Namensservice - eine Zuordnung von Namen zu Objekten, damit die Objekte über ihren Namen gefunden werden können. Was immer Sie an einem Rechner tun, Sie werden fast immer irgend welche Objekte manipulieren. Beispielsweise wenn Sie ein e-mail versenden: der Empfänger ist ein Objekt, Sie geben dessen Namen als Adresse ein; wenn Sie ein Datei verwenden: Sie greifen auf die Datei zu, indem Sie deren Namen verwenden.

Namensdienste verwenden also immer Namen, um Objekte nachzuschlagen.

Beispielsweise verwendet DNS Namen (www.xyz.com) für IP Adressen (123.123.123.123).

1.3.1.1. Namen

Was verstehen wir im Zusammenhang mit Namensdiensten unter einem *Namen*?
Der Namensdienst legt die Syntax fest, der ein gültiger Namen gehorchen muss. Diese Syntax bezeichnet man auch als *Namenskonvention.*

In der Microsoft Welt besteht beispielsweise ein Dateinamen aus der Laufwerksangabe plus einem Doppelpunkt und Verzeichnisangaben, jeweils getrennt durch '\' (back slash). Bei Unix ist die Namenskonvention anders (das Trennzeichen ist beispielsweise '/' (slash)).

Beim DNS System sieht die Namenskonvention völlig anders aus:
dort werden die einzelnen Bereiche von rechts nach links aufgelistet, durch '.' getrennt.
www.xyz.com ist eine Domäne, die zur Oberdomäne 'com' gehört.

Beim LDAP (Lightweight Directory Access Protocol) besteht ein Namen aus Komponenten, welche durch ein ',' getrennt werden und jeweils aus Name/Wert Paaren bestehen:
cn=Britta, o=KSZH, c=CH.

1.3.1.2. Bindings

Die Zuordnung eines Namens zu einem Objekt wird als Bindung, *binding* bezeichnet.

Beispiele:
1) der Dateiname ist an die Datei gebunden.
2) DNS bindet einen Maschinennamen an eine IP Adresse
3) ein LDAP Name wird an einen LDAP Verzeichniseintrag gebunden

1.3.1.3. Referenzen und Adressen

Je nach Namensservice kann ein Objekt nicht direkt darin abgespeichert werden. In diesem Fall speichert man eine Referenz auf das Objekt im Verzeichnis, es wird also ein Pointer oder eine *Referenz* im Namensservice abgespeichert. Die Referenz enthält die Information darüber, wie auf das Objekt zugegriffen werden kann. Das Objekt selber ist in diesem Fall in der Regel

komplex und enthält eventuell Zustandsinformationen. Die Referenz ist wesentlich kompakter.

Beispiel:
1) auf eine Datei greift man typischerweise über eine Dateireferenz oder eine file handle zu.
2) ein Druckerobjekt kann Informationen über die Warteschlange und weitere Zustandsinformationen enthalten

Der Inhalt der Referenz wird als *Adresse* (zum Objekt) bezeichnet. Aber die Unterscheidung zwischen der Referenz und deren Inhalt wird oft nicht sauber durchgezogen.

1.3.1.4. Context

Ein *Context* besteht aus einer Menge von Namen-zu-Objekt Bindungen. Zu jedem Context gehört eine Namenskonvention. Der Context stellt lookup Operationen (*Auflösungen* : welches Objekt gehört zu welchem Namen) zur Verfügung. Zudem stellt der Context auch Operationen zum Binden neuer Objekte, zum Ablösen (unbind) und dem Auflisten der Inhalte zur Verfügung.

Namen in einem Context können an mehrere Contexte gebunden sein, beispielsweise an *Subcontexte*, mit der selben Namenskonvention.

Beispiel:
1) jedes Verzeichnis des Windows Dateisystems kann auch Verzeichniseinträge haben, die Unterverzeichnisse.
2) bei DNS ist com ein Context, xyz.com ist ein Subcontext.

1.3.1.5. Namenssysteme und Namensräume

Ein *Namenssystem (naming system)* besteht aus einer zusammenhängenden Menge von Contexten des selben Typs (also gleicher Namenskonvention) und gemeinsamen Operationen (zum Beispiel bind, unbind, list).

Beispiele:
1) DNS ist ein Namenssystem
2) LDAP ist ein Namenssystem
3) das Windows Dateisystem ist ein Namenssystem

Ein Namenssystem stellt seinen Benutzern Dienste, *Namensdienste (naming services)*, zur Verfügung, mit deren Hilfe seine Kunden die benötigten Operationen durchführen können.

Beispiel:
1) DNS stellt einen Mechanismus zur Verfügung, mit dessen Hilfe zur IP Adresse der Name des Rechners bestimmt werden kann, und umgekehrt.
2) ein Dateisystem liefert zu einem Dateinamen die Datei oder das (sub)Verzeichnis.

Ein *Namensraum* (*namespace*) besteht aus der Menge der Namen in einem Namenssystem.

Beispiel:
1) Dateisystem: alle Dateinamen und Verzeichnisnamen zusammen bilden den Namensraum
2) LDAP: alle Einträge bilden den Namensraum

1.3.2. Verzeichniskonzepte

Viele Namensdienste werden durch *Verzeichnisdienste, directory services* ergänzt. Ein Verzeichnisdienst ordnet den Objekten Namen zu und gestattet es den Objekten *Attribute* zu haben. Sie können damit nicht nur Objekte nachschlagen, sondern auch deren Attribute bestimmen oder Objekte aufgrund ihrer Attribute suchen.

Beispiel:
1) Telefon: wenn Sie die Telefonnummer nachschlagen finden Sie in der Regel auch noch die Adresse oder den Beruf.
2) Dateisystem: neben der Datei erhalten Sie auch noch Informationen über die Grösse oder die letzten Änderungen (Datum).

Ein Verzeichnisobjekt stellt ein Objekt dar, beispielsweise einen Drucker, eine Person, einen Rechner, und enthält *Attribute*, welche das Objekt genauer beschreiben.

1.3.2.1. Attribute

Ein Verzeichnis kann *Attribute* haben. Beispiele kennen Sie aus Ihrer täglichen Arbeit mit dem Rechner:
1) der Drucker zeigt Ihnen in der Regel an, ob gedruckt wird und wieviele Aufträge noch in der Warteschlange sind.
2) das Dateisystem zeigt Ihnen global wieviel Speicherplatz benutzt und noch frei ist.

Ein Attribut besteht aus zwei Teilen, dem *Attributnamen* und dem *Attributwert*.

Beispiel:
Attributname: mailadresse; Attributwert : joller@joller-voss.ch

1.3.2.2. Verzeichnisse und Verzeichnisdienste

Ein *Verzeichnis* besteht aus einer zusammenhängenden Menge von Verzeichnisobjekten. Ein *Verzeichnisdienst* ist ein Dienst, der Operationen zur Verfügung stellt, um Attribute den Objekten im Verzeichnis zuzuordnen, zu löschen zu modifizieren oder was auch immer.

Der Novell Directory Service (NDS) ist ein Verzeichnisdienst, der von Novell entwickelt und von vielen Anbietern mitverwendet wird. Auf Solaris gibt es einen Network Information Service (NIS) und daneben existieren viele weitere Verzeichnisdienste, beispielsweise LDAP.

Sie finden ein Objekt in einem Verzeichnisdienst, indem Sie mit dem Namen des Objekts eine der Methoden aufrufen, die der Service zur Verfügung stellt. Viele Verzeichnisdienste stellen spezielle Suchmechanismen, *searches*, zur Verfügung. Die eigentliche Abfrage wird als *Suchfilter, search filter* bezeichnet. Den Suchvorgang bezeichnet man auch als *reverse lookup* oder *content-based searching*. Der Verzeichnisdienst sucht die Objekte und liefert jene zurück, welche dem Suchfilter genügen.

1.3.2.3. Kombinieren von Namens- und Verzeichnisdiensten

Verzeichnisse ordnen ihre Objekte oft hierarchisch an, in Baumstrukturen. Beispielsweise im LDAP wird ein sogenannter *directory information tree* (DIT) angelegt. In diesem Baum befinden sich beispielsweise Gruppen von Objekten, gruppiert nach bestimmten Kriterien.

1.4. Typische Probleme

1.4.1. Übersetzungsprobleme

Als Hilfe zur Behebung der üblichen Probleme, hier eine Liste möglicher Fehler und möglicher Korrekturaktionen.

1.4.1.1. Class or Package Not Found

Problem:
Sie erhalten die Meldung "Package javax.naming not found" oder eine ähnliche Fehlermeldung über fehlende Klassen.

Ursache:
Sie haben die JNDI Klassen (jndi.jar) nicht im Klassenpfad CLASSPATH
oder
sie haben die JNDI Klassen nicht korrekt installiert.

Lösung:
Java 2 SDK, v1.3 umfasst die JNDI Klassen standardmässig. Falls Sie eine ältere Version verwenden, müssen Sie die Klassen, bzw. jar Dateien separat installieren.

Sie finden diese Dateien inklusive verschiedenen Providerklassen auf dem Server.

Falls Sie Java 2 SDK, v1.2 verwenden, müssen Sie jndi.jar im Verzeichnis JAVA_HOME/jre/lib/ext speichern. JAVA_HOME ist zum Beispiel c:\jdk1.2.2 ohne abschliessenden Backslash.

In JDK 1.1 muss das Verzeichnis von jndi.jar entweder in der CLASSPATH Umgebungsvariable vorkommen, oder mit der -classpath Option gesetzt werden.

1.4.1.2. Incompatible Java Platform Versions

Problem:
Sie erhalten eine Fehlermeldung, die anzeigt, dass entweder java.* Packages oder Klassen fehlen.

Ursache:
Sie verwenden eine zu alte Java Version.

Lösung:
Sie benötigen Java 1.1.2 oder höher : http://java.sun.com/products/jdk.

1.4.2. Laufzeit Probleme

Hier ist eine Liste der häufigsten Laufzeitfehler.

1.4.2.1. Class Not Found

Problem:
Sie erhalten einen NoClassDefFoundError beim Starten des Programms.

Ursache:
Sie haben die JNDI Klassen (jndi.jar) nicht im Klassenpfad oder haben die JNDI Klassen nicht korrekt installiert.

Solution:
Java 2 SDK, v1.3 umfasst bereits die JNDI Klassen Falls Sie Java 2 SDK, v1.3, dann sollten also die Klassen bereits vorhanden sein.

Falls Sie Java 2 SDK, v1.2 verwenden, müssen Sie die Umgebungsvariable JAVA_HOME definieren (c:\jdk1.2.2)und sicher sein, dass jndi.jar im JAVA_HOME/jre/lib/ext Verzeichnis vorhanden ist

In JDK 1.1 muss der Klassenpfad CLASSPATH angepasst werden

1.4.2.2. No Initial Context

Problem:
NoInitialContextException.

Ursache:
Die Initial Context Factory wurde nicht spezifiziert, spezifisch: die INITIAL_CONTEXT_FACTORY Umgebungseigenschaft

Lösung:
Sie müssen die INITIAL_CONTEXT_FACTORY Umgebungsvariable setzen.

1.4.2.3. Connection Refused

Problem:
Sie erhalten eine CommunicationException: "connection refused."

Ursache:
Der Server und Port aus dem Context.PROVIDER_URL wird nicht bedient.

Lösung:
Prüfen Sie den Server und den Port.

1.4.2.4. Connection Fails

Problem:
Der LDAP Server antwortet nicht.

Ursache:
unter Umständen antwortet der (public) Server nicht korrekt auf eine LDAP v3 Anfrage-

Lösung:
Versuchen Sie es mit Version 2 : "java.naming.ldap.version = 2".

oder im Anwendungsprogramm:

```
env.put(Context.REFERRAL, "throw");
```

1.4.2.5. Programm hängt sich auf

Problem:
Programm hängt sich auf

Ursache:
Einige (vorallem public Server) antworten nicht, falls man zu allgemeine Resultate abfragen möchte oder Abfragen zu oft durchführt.

Oder, falls Sie versuchen mit Secure Socket Layer (SSL) mit einem Server zu kommunizieren, der SSL nicht unterstützt, dann trit dieser Fehler auch auf.

Lösung:
Im Falle vom SSL Fehler können Sie mit der Umgebungsvariable SECURITY_PROTOCOL das Problem beseitigen.

1.4.2.6. Name Not Found

Problem:
Sie erhalten die Fehlermeldung NameNotFoundException.

Ursache:
Context Fehler in LDAP;
Beispiel: Context.PROVIDER_URL sei "ldap://ldapserver:389/o=JNDIBeispiel" und ein Name "cn=Josef,c=us",

dann ist der gesamte, vollständige Namen aus Sicht vom LDAP Service "cn=Joe,c=us,o=JNDIBeispiel".

Lösung:
Prüfen Sie, ob der Namen auf dem Server existiert.

1.5. Vorbereitende Arbeiten für die Beispiele

Damit Sie die Beispiele nachvollziehen können, müssen Sie folgende Software installiert haben oder installieren:

1.5.1. Platform Software

Sie benötigen JDK 1.2+ oder 1.3, einen Browser (Navigator 4+, Explorer 5+) und vermutlich das Java Plugin, falls Sie mit Applets arbeiten möchten.

1.5.2. JNDI Software

Falls Sie JDK1.3 installiert haben, dann müssen Sie nichts installieren. Sonst sollten Sie die Klassen von JNDI und verschiedene Service Provider Klassen installieren. Diese befinden sich auf dem Server unter ...JDNI_Klassen ...
Natürlich können Sie auch die neusten Versionen ab dem Java Web herunterladen: http://java.sun.com/products/jndi .

1.5.3. Service Provider Software

Das JNDI API ist ein generisches API für den Zugriff auf Namens- und Verzeichnisservices. Die Architektur gestattet es, unterschiedliche Serviceprovider einzuschieben (pluggable API).

Der Serviceprovider ist die Software, welche das JNDI API auf aktuelle Namens- und Verzeichnis- Server abbildet. JNDI Clients und der JNDI Provider sind Clients, der Namens/Verzeichnisserver ist der Server.

JDK 1.3 oder die oben erwähnten Klassen, gestatten einem Client auf LDAP, COS (CORBA) oder RMI Registry Verzeichnisse / Registries zuzugreifen.

Im Folgenden benutzen wir:

- das Dateisystem als Serviceprovider oder
- LDAP Service Provider

Wie bereits erwähnt, kann ein LDAP Server gratis ab http://www.OpenLDAP.org herunter geladen werden. Mehrere public LDAP Provider wurden ebenfalls bereits erwähnt:

- `ldap://ldap.Bigfoot.com`
- `ldap://ldap.four11.com`
- `ldap://ldap.InfoSpace.com`
- `ldap://ldap.yahoo.com`

1.5.4. Inhalt des Verzeichnisses

Falls ein Verzeichnis bereits existiert, und dies ist bei den obigen freien ldap://ldap... der Fall, welche Informationen können Sie dann mit einem Anwendungsprogramm daraus heraus holen?

Grundsätzlich enthält jedes Verzeichnis "bindings" und Attribute, wobei wir Bindings bereits besprochen haben (Name-Objekt Paare). Jedes Objekt im Verzeichnis besitzt einen Namen. Man greift auf dieses Objekt zu, indem man den Namen als Parameter einer Zugriffsroutine verwendet.

Neben den Objekten werden auch Attribute abgespeichert. Attribute sind optional für die Objekte. Man kann das Verzeichnis nach Objekten mit bestimmten Attributen oder aber ein Objekt nach dessen Attribute abfragen.

1.5.4.1. Verzeichnisschemata

Ein Schema spezifiziert den Typus der Objekte, die ein Verzeichnis aufnehmen kann. In der Praxis geht man folgendermassen vor:

- entweder man fügt ein Schema zum Verzeichnis. Damit kann man lediglich bestimmte, eben Schema- konforme Objekte in diesem Verzeichnis abspeichern
- oder man lässt das Schema weg, speichert die Objekte (ohne Schema- Checking) und fügt anschliessend das Schema hinzu. Da die Objekte bereits gespeichert sind, bleiben Sie im Verzeichnis.

Das Format eines Schemas wurde im RFC 2252 festgelegt. Hier als Beispiel ein Schema für Java Objekte:

```
-- Attribute types --

( 1.3.6.1.4.1.42.2.27.4.1.6
  NAME 'javaClassName'
  DESC 'Fully qualified name of distinguished Java class or interface'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE
)

( 1.3.6.1.4.1.42.2.27.4.1.7
  NAME 'javaCodebase'
  DESC 'URL(s) specifying the location of class definition'
  EQUALITY caseExactIA5Match
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
)

( 1.3.6.1.4.1.42.2.27.4.1.8
  NAME 'javaSerializedData'
  DESC 'Serialized form of a Java object'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
  SINGLE-VALUE
)

( 1.3.6.1.4.1.42.2.27.4.1.10
  NAME 'javaFactory'
  DESC 'Fully qualified Java class name of a JNDI object factory'
```

```
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE
)

( 1.3.6.1.4.1.42.2.27.4.1.11
  NAME 'javaReferenceAddress'
  DESC 'Addresses associated with a JNDI Reference'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)

( 1.3.6.1.4.1.42.2.27.4.1.12
  NAME 'javaDoc'
  DESC 'The Java documentation for the class'
  EQUALITY caseExactIA5Match
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
)

( 1.3.6.1.4.1.42.2.27.4.1.13
  NAME 'javaClassNames'
  DESC 'Fully qualified Java class or interface name'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)

-- from RFC-2256 --

( 2.5.4.13
  NAME 'description'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{1024}
)

-- Object classes --

( 1.3.6.1.4.1.42.2.27.4.2.1
  NAME 'javaContainer'
  DESC 'Container for a Java object'
  SUP top
  STRUCTURAL
  MUST ( cn )
)

( 1.3.6.1.4.1.42.2.27.4.2.4
  NAME 'javaObject'
  DESC 'Java object representation'
  SUP top
  ABSTRACT
  MUST ( javaClassName )
  MAY ( javaClassNames $ javaCodebase $ javaDoc $ description )
)

( 1.3.6.1.4.1.42.2.27.4.2.5
  NAME 'javaSerializedObject'
  DESC 'Java serialized object'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)
```

```
( 1.3.6.1.4.1.42.2.27.4.2.7
  NAME 'javaNamingReference'
  DESC 'JNDI reference'
  SUP javaObject
  AUXILIARY
  MAY ( javaReferenceAddress $ javaFactory )
)

( 1.3.6.1.4.1.42.2.27.4.2.8
  NAME 'javaMarshalledObject'
  DESC 'Java marshalled object'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)

-- Matching rule from ISO X.520 --

( 2.5.13.5
  NAME 'caseExactMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)
```

Je nach Server können diese Dateien unterstützt erstellt werden, Sie brauchen sich also nicht um alle Details zu kümmern.

Zudem stehen Java Programme zur Verfügung, mit denen diese Dateien kreiert werden können. Diese stammen von Sun Microsystems.

Sie finden auf dem Server die Java Programme zum Generieren des Java, des CORBA Schemas und entsprechende Update- Programme.

- `CreateJavaScheme.java`
- `CreateCORBASchema.java`
- `UpdateJavaScheme.java`
- `UpdateCORBAScheme.java`

Im dazugehörigen `ReadMe.txt` finden Sie auch eine Erklärung der Codierung in der obigen Darstellung.

Falls man den Netscape Directory Server verwendet, muss man in dessen Konfigurationsdatei `java-object-schema.conf` mit dem generierten Inhalt des Ergebnisses des Updateprogramms modifizieren.

1.5.4.2. Verzeichnisse für die Beispiele

Sie können das Programm im Verzeichnis "`SetupFuerBeispiele`" ausführen, um die Verzeichnisse für die Beispiele anzulegen.

Der Programmaufruf beinhaltet einen Aufrufparameter : das <root> Verzeichnis. Dieses kann, muss aber noch nicht existieren. Ein weiterer Parameter gibt an, ob die Verzeichnisse gelöscht werden sollen.

1.5.4.3. Packages und CLASSPATH

Um JNDI Programme erfolgreich übersetzen zu können, müssen verschiedene Packages (jeweils nicht alle) importiert werden.

Import der JNDI Klassen

Die JNDI Packages

- `java.naming`
- `java.naming.directory`
- `java.naming.event`
- `java.naming.ldap`
- `java.naming.spi`

müssen ganz oder teilweise importiert werden, jeweils je nach Anwendung.

Übersetzen der Programme

Falls man JDK1.3 verwendet, sind die entsprechenden Klassen bereits automatisch vorhanden. Sonst muss entsprechend den Anweisungen weiter vorne vorgegangen werden:

- falls JDK1.2 installiert ist, muss JAVA_HOME (=..jdk1.2) definiert werden
- falls JDK1.1 installiert ist, muss der CLASSPATH entsprechend ergänzt werden.

Die betrifft jeweils das jndi.jar Archiv.

Ausführen der Programme

Auch beim Ausführen der Programme müssen wie oben die Umgebungsvariablen (JAVA_HOME, CLASSPATH) gesetzt werden, und dort die entsprechenden Archive gefunden werden.

1.5.5. Der InitialContext

Im einzelnen müssen folgende Schritte durchlaufen werden:

1. es muss ein Service Provider ausgewählt werden, entsprechend dem gewünschten Service
2. die Konfigurationen müssen angepasst werden
3. der `InitalContext` Konstruktor muss im Programm aufgerufen werden

1.5.5.1. Auswahl des Service Providers für den InitialContext

Man kann den Serviceprovider dadurch spezifizieren, indem man ein Set von Environment Properties (eine Hashtabelle) kreiert und den Namen der Serviceproviderklasse hinzufügen.

Programmfragment:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.jndi.ldap.LdapCtxFactory");
```

Falls man das Dateisystem als Verzeichnis verwenden möchte, kann man die von Sun zur Verfügung gestellte Factory verwenden:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.jndi.fscontext.Ref FSContextFactory");
```

1.5.5.2. Informationen über den InitialContext ergänzen

Je nach Verzeichnisdienst müssen unterschiedliche Angaben ergänzt werden. Dies geschieht mit Hilfe von "Environment Properties" also Umgebungsvariablen und Eigenschaften.

Programmfragment:

```
env.put(Context.PROVIDER_URL, "ldap://ldap.yahoo.com:389");
env.put(Context.SECURITY_PRINCIPAL, "jndibeispiele");
```

In den Beispielen verwenden wir das Dateisystem und den LDAP Server als Service Provider. Analog zum obigen Programmfragment für den LDAP Server hätten wir im Falle des Dateisystems:

```
env.put(Context.PROVIDER_URL, "file:/JNDI/Beispiele");
// ACHTUNG : file:/ nicht file://
```

Falls wir einen lokalen LDAP Server verwenden, an Port 389 des localhost und unser Root "Beispiele" ist, dann würde die obige LDAP Zeile wie folgt aussehen:

```
env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=Beispiele");
```

(o=... steht für Organisation ... gemäss der LDAP Syntax)

1.5.5.3. Kreieren des InitialContext

Nun sind wir bereit, den InitalContext unserer Anwendung zu kreieren:

```
Context ctx = new InitialContext(env);
// Context ist ein Interface; InitialContext eine Klasse
```

Damit können wir auf Namensdienste zugreifen.
Um auch Verzeichnisdienste benutzen zu können, müssen wir einen DirContext kreieren:

```
DirContext ctx = new InitialDirContext(env);
```

1.5.6. Namen

Die Klassen `InitialDirContext` und `InitialContext` Klassen verwenden Methoden, die entweder als Parameter `Name` oder `String` Variablen aufnehmen. Ein Name muss dabei der entsprechenden Syntax des Namensraumes gehorchen. Was im Detail darunter zu verstehen ist, werden wir gleich konkreter sehen.

1.5.7. Naming Operationen

In den folgenden Abschnitten befassen wir uns mit den Möglichkeiten eines Namensraumes:

- nachschauen eines Objektes
- Auflisten des Inhaltes eines Contextes
- hinzufügen, überschreiben oder entfernen einer Bindung
- umbenennen einer Bindung
- kreieren und löschen von Subkontexten

Für die folgenden Beispiele verwenden wir das Dateisystem. Sie müssen aber die Verzeichnisse des Namensraumes mit dem Setup Programm kreieren, wenn Sie die Beispiele testen möchten.

1.5.7.1. Nachschauen eines Objektes

Diese Lookup Operation verwendet die Methode Context.lookup(). Als Parameter muss der Name des Objekts angegeben werden.

```
package lookupns;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * look up eines object.
  *
  * usage: java Lookup
  */
public class Lookup {
    public static void main(String[] args) {

      // kreiere initial context
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");

      try {
          // kreiere initial context
          Context ctx = new InitialContext(env);

          // lookup und cast
          File f = (File) ctx.lookup("report.txt");

          System.out.println("Lookup erfolgreich: "+f);

          // Close
```

```
            ctx.close();
          } catch (NamingException e) {
            System.out.println("Lookup fehlgeschlagen: " + e);
          }
        }
}
```

Als Ausgabe erhalten Sie je nach dem ob sich die Datei "report.txt" im obigen Verzeichnis befindet oder nicht eine entsprechende Ausgabe.

Falls Sie den Verzeichnisbaum mit dem Hilfsprogramm "`SetupFuerBeispiele`" generiert haben und dieses Verzeichnis im Root steht, sollten Sie die Datei finden:

```
Lookup erfolgreich: D:\Test\report.txt
```

1.5.7.2. Auflisten eines Contextes

Wir können auch den gesamten Inhalt des Kontextes nachschauen, anstatt nur ein Objekt. Dafür stehen unterschiedliche Methoden zur Verfügung:

- `Context.list()`
- `Context.listBindings()`

Die Context.list Methode

Als erstes betrachten wir ein `list()` Beispiel:

```
package listebsp;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

import javax.naming.*;
import java.util.Hashtable;

/**
 * Demo: auflisten des Namens und der Klasse der Objekte im Context
 *
 * usage: java Liste
 */
public class Liste {
    public static void main(String[] args) {

    // Achtung : Sie müssen zuerst das Setup Programm laufen lassen!!

    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.sun.jndi.fscontext.RefFSContextFactory");
    // Verzeichnis muss angepasst werden: Root der generierten
    // Verzeichnisse
    env.put(Context.PROVIDER_URL, "file:/Test");
    try {
      Context ctx = new InitialContext(env);
```

```
      NamingEnumeration list = ctx.list("awt");
      while (list.hasMore()) {
        NameClassPair nc = (NameClassPair)list.next();
      System.out.println(nc);
      }
      ctx.close();
    } catch (NamingException e) {
      System.out.println("List fehlgeschlagen: " + e);
    }
  }
}
```

Mit der Ausgabe (nachdem das Setup Programm ausgeführt wurde)

```
accessibility: javax.naming.Context
color: javax.naming.Context
datatransfer: javax.naming.Context
dnd: javax.naming.Context
event: javax.naming.Context
font: javax.naming.Context
geom: javax.naming.Context
im: javax.naming.Context
image: javax.naming.Context
peer: javax.naming.Context
print: javax.naming.Context
swing: javax.naming.Context
```

1.5.7.3. Auflisten der Bindings

Die Methode `Context.listBindings()` liefert eine Liste der Bindings im Namensraum.

```
package listbindingsbsp;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

import javax.naming.*;
import java.util.Hashtable;

/**
 * Demo : List Bindings im Context.
 *
 * usage: java ListBindings
 */
public class ListBindings {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");
      try {
          Context ctx = new InitialContext(env);
          NamingEnumeration bindings = ctx.listBindings("awt");
          while (bindings.hasMore()) {
            Binding bd = (Binding)bindings.next();
            System.out.println(bd.getName() + ": " + bd.getObject());
          }
          ctx.close();
```

```
      } catch (NamingException e) {
          System.out.println("List Bindings fehlgeschlagen: " + e);
      }
    }
}
```

Und hier die Ausgabe (Bindings verweisen auf Objekte):

```
accessibility: com.sun.jndi.fscontext.RefFSContext@b9ac41a
color: com.sun.jndi.fscontext.RefFSContext@f2ac41a
datatransfer: com.sun.jndi.fscontext.RefFSContext@72c41a
dnd: com.sun.jndi.fscontext.RefFSContext@54ac41a
event: com.sun.jndi.fscontext.RefFSContext@19d2c41a
font: com.sun.jndi.fscontext.RefFSContext@132ec41a
geom: com.sun.jndi.fscontext.RefFSContext@16eec41a
im: com.sun.jndi.fscontext.RefFSContext@1592c41a
image: com.sun.jndi.fscontext.RefFSContext@2886c41a
peer: com.sun.jndi.fscontext.RefFSContext@2e5ec41a
print: com.sun.jndi.fscontext.RefFSContext@2df2c41a
swing: com.sun.jndi.fscontext.RefFSContext@20cec41a
```

1.5.7.4. Abbruch der Auflistung

Die Auflistung der Bindings oder Namens-Objekt Paare kann auf unterschiedliche Weise limitiert werden.

- durch Einsatz der `NamingEnumeration.hasMore()` Methode und schrittweises Abarbeiten
- durch Schliessen der Liste mit `NamingEnumeration.close()` und
- durch Einsatz und manuelles Zählen der Einträge mit `hasMore()` und `next()` Methoden

1.5.7.5. Warum gibt es zwei so ähnliche Methoden?

Es gibt verschiedene Gründe für die `list()` und `listBindings()` Methoden:

1. `list()`:
   liefert Browser ähnlich eine Liste, die man anzeigen kann
2. `listBindings()`:
   ist eher für Applikationen bestimmt, die auf Objekte zugreifen müssen; daher ist die Bindung entscheidend. Die Methode ist aufwendiger als `list()`.

1.5.8. Bindungen hinzufügen, ersetzen und entfernen

Das Context Interface enthält Methoden, mit denen Bindungen entfernt, hinzugefügt oder ersetzt werden können.

1.5.8.1. Hinzufügen einer Bindung

In diesem Beispiel (FruchtBsp) kreieren wir mit Hilfe einer Früchtefabrik (`FruitFactory`) Fruchtobjekte (`Fruit`). Diese werden gebunden mit `Context.bind()`.
Diese Operation legt eine Datei `.bindings` an. Wenn das Programm zweimal gestartet wird, bricht es das zweite Mal wegen zweifacher Bindung mit Fehler ab (`rebind` löst das Problem). Sie können aber auch einfach die Datei löschen.

```
package fruchtbsp;
```

```
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * Demo: Bindung an einen Context hinzufügen
  *
  * usage: java Bind
  */

public class Bind {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");
      try {
          Context ctx = new InitialContext(env);
          Fruit fruit = new Fruit("Orange");
          ctx.bind("lieblingsfrucht", fruit);
          Object obj = ctx.lookup("lieblingsfrucht");
          System.out.println(obj);

          // Close the context when we're done
          ctx.close();
      } catch (NamingException e) {
          System.out.println("Operation failed: " + e);
      }
    }
}
```

Die Frucht wird in folgender Klasse definiert:

```
package fruchtbsp;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
import javax.naming.*;
public class Fruit implements Referenceable {
    String fruit;

    public Fruit(String f) {
      fruit = f;
    }

    public Reference getReference() throws NamingException {
      return new Reference(
          Fruit.class.getName(),
```

```
            new StringRefAddr("fruit", fruit),
            FruitFactory.class.getName(),
            null);          // factory
    }

    public String toString() {
      return fruit;
    }
}
```

und die Fruchtfabrik:

```
package fruchtbsp;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */


import javax.naming.*;
import javax.naming.spi.ObjectFactory;
import java.util.Hashtable;

public class FruitFactory implements ObjectFactory {
    public FruitFactory() {
    }

    public Object getObjectInstance(Object obj, Name name, Context ctx,
      Hashtable env) throws Exception {
      if (obj instanceof Reference) {
          Reference ref = (Reference)obj;
          if (ref.getClassName().equals(Fruit.class.getName())) {
            RefAddr addr = ref.get("fruit");
            if (addr != null) {
                return new Fruit((String)addr.getContent());
            }
          }
      }
      return null;
    }
}
```

Ausgabe bei zwei Läufen:

```
Orange
Operation failed: javax.naming.NameAlreadyBoundException: lieblingsfrucht
```

1.5.8.2. Ersetzen einer Bindung

Mit `Context.rebind()` kann eine Bindung ersetzt werden.
package fruchtrebind;

```
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * Demo: überschreiben einer Bindung
  *
  * usage: java Rebind
  */
public class Rebind {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");

      try {
          Context ctx = new InitialContext(env);
          Fruit fruit = new Fruit("Starfrucht");

          // Perform the bind
          ctx.rebind("favorite", fruit);

          // Check that it is bound
          Object obj = ctx.lookup("favorite");
          System.out.println(obj);

          // Close the context when we're done
          ctx.close();
      } catch (NamingException e) {
          System.out.println("Operation failed: " + e);
      }
    }
}
```

Die Ausgabe ist einfach der neue Früchtenamen, hier also "Starfrucht".

1.5.8.3. Entfernen einer Bindung

Dazu verwenden wir die Methode `Context.unbind`() : (auch hier haben wir es mit Früchten zu tun).

Achtung: `unbind()` **kann mehrfach ausgeführt werden, ohne Exception**
Das heisst, das Entfernen eines Objektes mit `unbind()` zeigt nicht genau auf, was wirklich geschieht.

```
package unbindns;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * Demo: unbind
  *
  * usage: java Unbind
  */
public class Unbind {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");
      try {
          Context ctx = new InitialContext(env);
          ctx.unbind("Lieblingsfrucht");
          Object obj = null;
          try {
            obj = ctx.lookup("Lieblingsfrucht");
          } catch (NameNotFoundException ne) {
            System.out.println("unbind erfolgreich");
            return;
          }
          System.out.println("unbind fehlgeschlagen; Objekt immer noch
gebunden: " + obj);
          ctx.close();
      } catch (NamingException e) {
          System.out.println("Operation fehlgeschlagen: " + e);
      }
    }
}
```

Mit der Ausgabe:
unbind erfolgreich

1.5.9. Umbenennen eines Objektes

Mit Hilfe der Methode `Context.rename()` kann ein Objekt umbenannt werden:

```
package renamens;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * Demo: rename eines gebundenen Objekts
  *
  * usage: java Rename
  */

public class Rename {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");
      try {
          Context ctx = new InitialContext(env);
          Object obj = ctx.lookup("report.txt");
          System.out.println("Alter Objektname:"+obj);

          ctx.rename("report.txt", "old_report.txt");
          obj = ctx.lookup("old_report.txt");
          System.out.println("Neuer Objektname:"+obj);
          ctx.rename("old_report.txt", "report.txt");
          obj = ctx.lookup("report.txt");
          System.out.println("und wieder neuer Objektname:"+obj);
          ctx.close();
      } catch (NamingException e) {
          System.out.println("Rename fehlgeschlagen: " + e);
      }
    }
}
```

Das Programm liefert die Ausgabe:

```
Alter Objektname:D:\JavaNameingDirectoryServices\Beispiele\report.txt
Neuer Objektname:D:\JavaNameingDirectoryServices\Beispiele\old_report.txt
und wieder neuer Objektname:D:\JavaNameingDirectoryServices\Beispiele\report.txt
```

1.5.10. Kreieren und Zerstören eines Contexts

Wir wollen nicht gleich den gesamten Context zerstören. Aber einen Subkontext können wir ja mal manipulieren:

package createsubcontextfs;

```
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
```

```
import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * Demo: kreiere Subkontext "new".
  *
  * usage: java Create
  */
public class Create {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:/Test");
      try {
        System.out.println("Kreieren des InitialContext");
      Context ctx = new InitialContext(env);
        System.out.println("Kreieren des Subcontext");
      Context result = ctx.createSubcontext("new");
        System.out.println("Pruefen des Subcontext - Parent");
      NamingEnumeration list = ctx.list("");
        System.out.println("Pruefen des Context");
        while (list.hasMore()) {
            NameClassPair nc = (NameClassPair)list.next();
            System.out.println(nc);
          }
        ctx.close();
      } catch (NamingException e) {
          System.out.println("Create fehlgeschlagen: " + e);
      }
    }
}
```

Das Programm liefert folgende Ausgabe:

```
Kreieren des InitialContext
Kreieren des Subcontext
Pruefen des Subcontext - Parent
Pruefen des Context
awt: javax.naming.Context
createSubcontextFS: javax.naming.Context
FruchtBsp: javax.naming.Context
```

```
FruchtRebind: javax.naming.Context
java: javax.naming.Context
LesenDerAttribute: javax.naming.Context
listBindingsBsp: javax.naming.Context
ListeBsp: javax.naming.Context
LookupBsp: javax.naming.Context
LookupNS: javax.naming.Context
new: javax.naming.Context
renameNS: javax.naming.Context
report.txt: java.io.File
UnbindNS: javax.naming.Context
```

Das Verzeichnis "new" ist natürlich noch leer.

1.5.10.1. Zerstören eines Subcontextes

Nachdem wir einen neuen Subcontext kreiert haben, wollen wir auch einen zerstören.

```
package destroySubcontextNS;

import javax.naming.*;
import java.io.File;
import java.util.Hashtable;

/**
  * Demo: zerstört Subkontext "new".
  *
  * usage: java Destroy
  */
class Destroy {
    public static void main(String[] args) {
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
                 env.put(Context.PROVIDER_URL, "file:/Test");

      try {
          Context ctx = new InitialContext(env);
          ctx.destroySubcontext("new");
          NamingEnumeration list = ctx.list("");
          while (list.hasMore()) {
            NameClassPair nc = (NameClassPair)list.next();
            System.out.println(nc);
          }
          ctx.close();
      } catch (NamingException e) {
          System.out.println("destroy fehlgeschlagen: " + e);
      }
    }
}
```

mit der Ausgabe:

```
awt: javax.naming.Context
createSubcontextFS: javax.naming.Context
destroySubcontextNS: javax.naming.Context
FruchtBsp: javax.naming.Context
FruchtRebind: javax.naming.Context
java: javax.naming.Context
LesenDerAttribute: javax.naming.Context
listBindingsBsp: javax.naming.Context
ListeBsp: javax.naming.Context
LookupBsp: javax.naming.Context
LookupNS: javax.naming.Context
renameNS: javax.naming.Context
report.txt: java.io.File
UnbindNS: javax.naming.Context
```

Offensichtlich fehlt hier das "new" Verzeichnis.

JAVA NAMING UND DIRECTORY INTERFACES