Java User Group München

Java & LDAP

Stefan Seelmann
Christine Koppelt

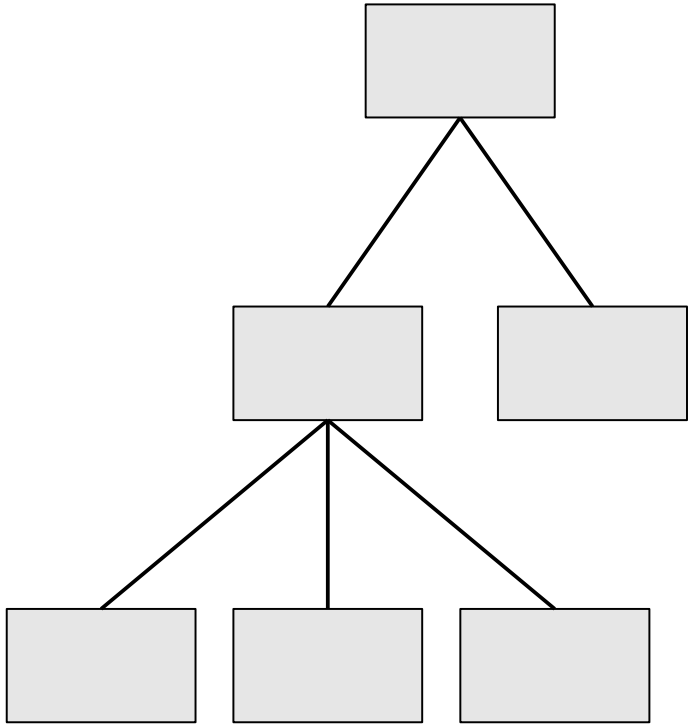
Überblick

- Grundlagen von LDAP
- JNDI (Java Naming & Directory Interface)
- Object-LDAP Mapping
- LDAP Server „embedded“
- LDAP & Web-Applikationsserver

Grundlagen von LDAP

Was ist LDAP?

- Lightweight Directory Access Protocol
  - Standardisiertes Protokoll zum Zugriff auf Verzeichnisse
  - Aktuelle Version LDAPv3
- Verzeichnisse
  - Speichern Daten in Form von Einträgen
  - Einträge sind in einer Baumstruktur angeordnet
- Client/Server Modell



Einsatzgebiete von LDAP

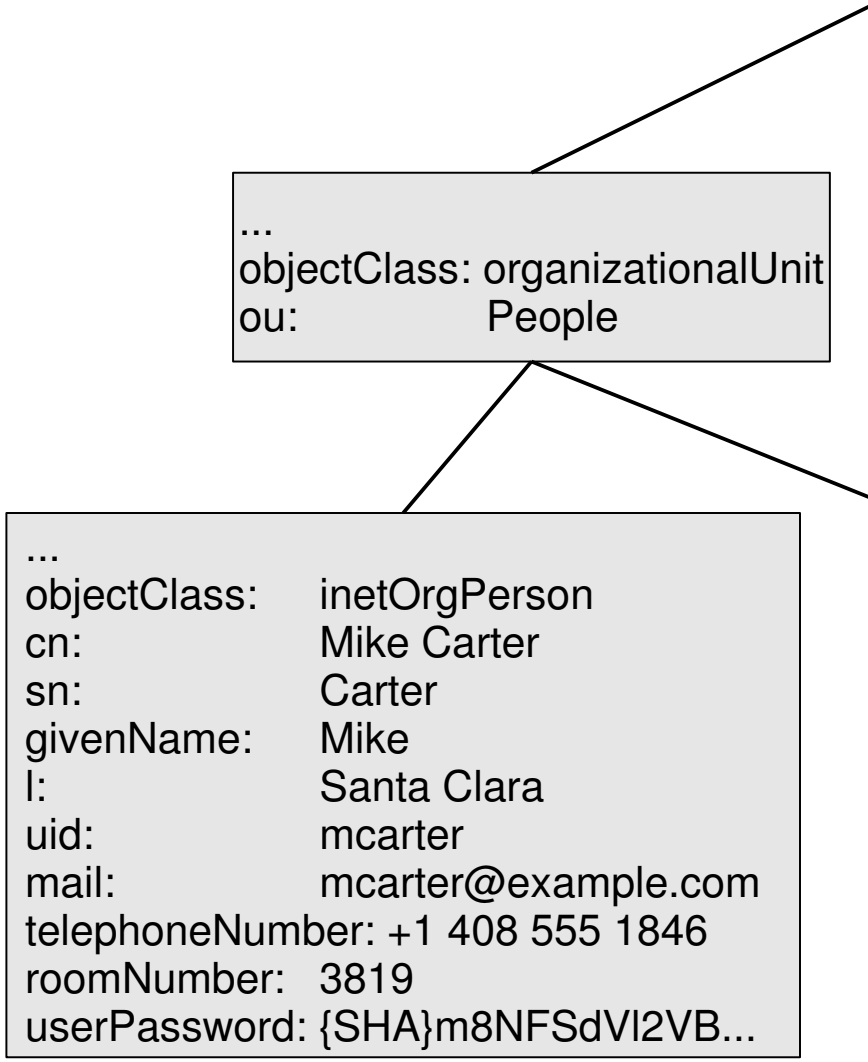
- Zentrale Verwaltung von Benutzern und deren Berechtigungen
  - Authentifizierung
  - Autorisierung
  - Serverlösungen, die Verzeichnisse integrieren
    - z.B. Mailserver, Webserver, Portalserver
- Zentrale Speicherung von Konfigurationsdaten
- Adressbuch/Mitarbeiterverzeichnis
  - z.B. für Mailclients

LDAP-Serverprodukte

- OpenLDAP
  - In den Repositories zahlreicher Linux-Distributionen enthalten
- Nachfolger des Netscape Directory Servers
  - Fedora/Red Hat Directory Server
  - Sun Java System Directory Server
- Zahlreiche weitere kommerzielle Produkte
  - Microsoft (Active Directory), Oracle, Novell, Siemens
- Java
  - ApacheDS
  - OpenDS

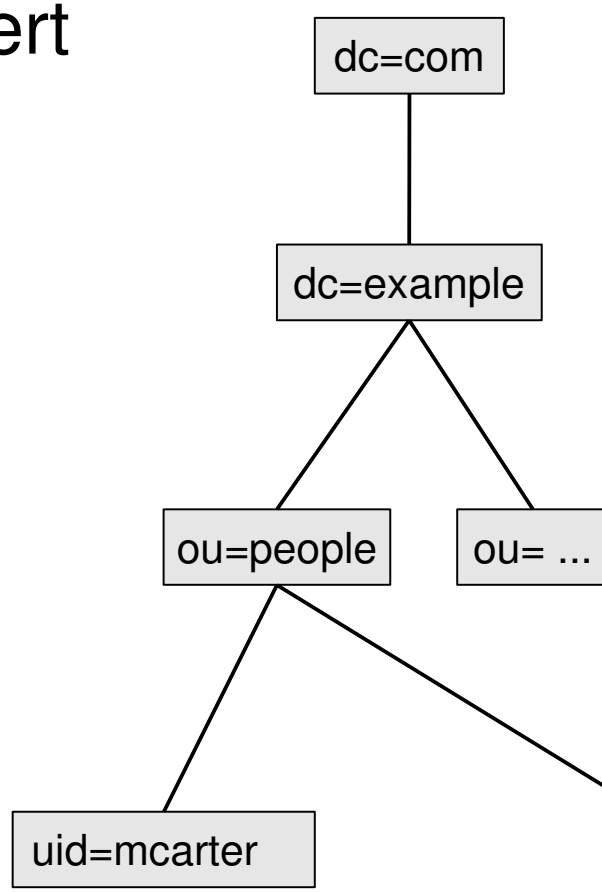
Informationsmodell

- Attribute
  - Attributnamen/Wertpaare
  - ein- oder mehrwertig
  - standardisiert
- Objektklassen
  - may/must Attribute
  - standardisiert
- Schema
  - Legt fest, welche Attribute und Objektklassen in einem Verzeichnis gespeichert werden dürfen



Identifikation von Einträgen

- DN (Distinguished Name) identifiziert Eintrag eindeutig
- DN setzt sich zusammen aus
  - RDN (Relative Distinguished Name)
    - ausgewähltes Attribut(e) des Eintrags
    - muss innerhalb der Geschwisterknoten eindeutig sein
  - Position innerhalb des Baumes
- DN ist nicht stabil
  - -> UUID für jeden Eintrag



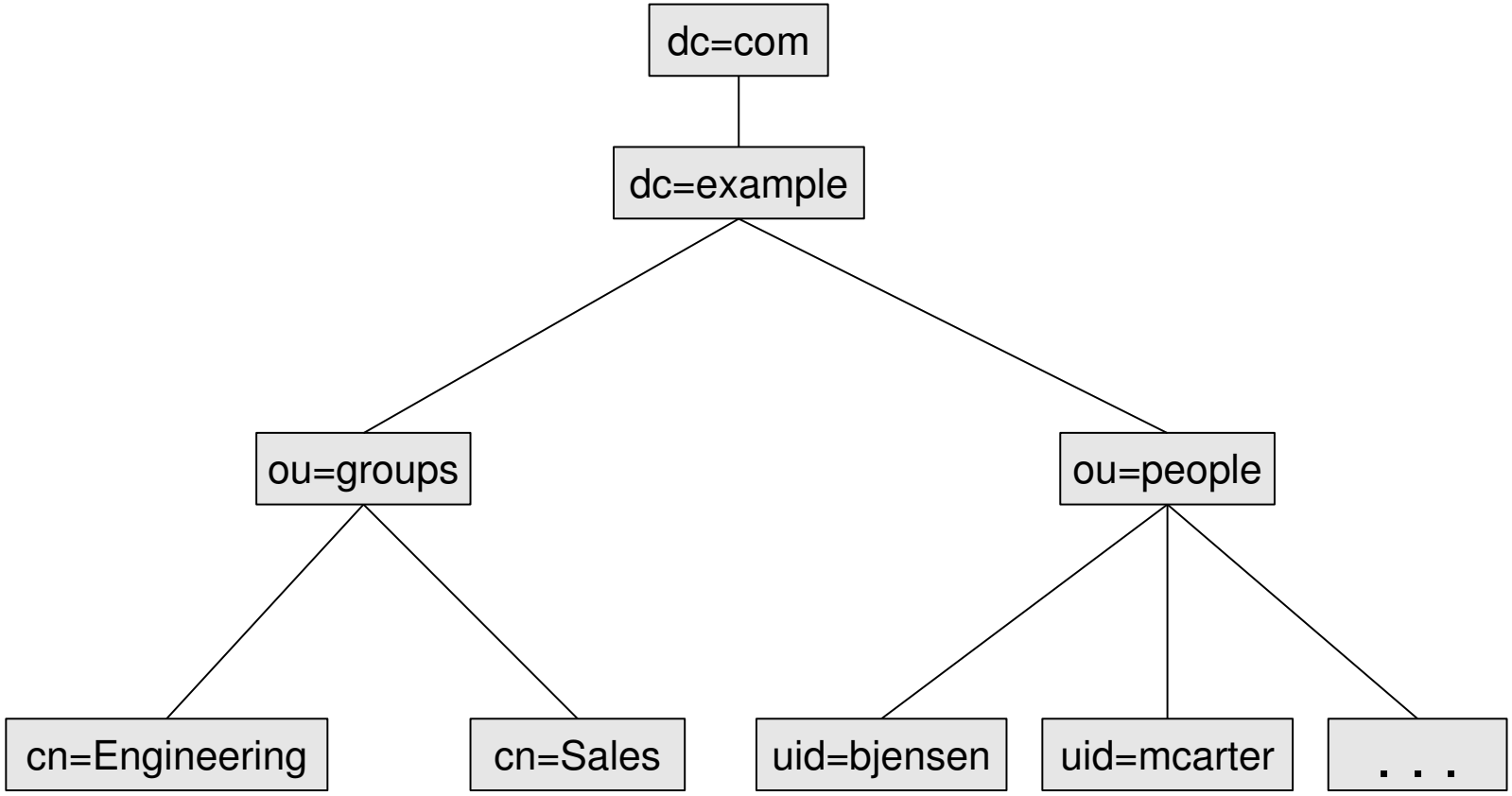
dn: uid=mcarter,ou=people,dc=example,dc=com

Gruppen

- Spezielle Einträge mit
  - der Objektklasse `groupOfNames` oder `groupOfUniqueNames`
  - einem Namen
  - einer Liste von DNs
- Achtung: LDAP kennt keine referentielle Integrität und keine Foreign Keys!

```
dn: cn=Engineering,ou=groups,dc=example,dc=com
objectClass: groupOfUniqueNames
cn: Engineering
uniqueMember: uid=bjensen,ou=people,dc=example,dc=com
uniqueMember: uid=speterson,ou=people,dc=example,dc=com
uniqueMember: uid=mwhite,ou=people,dc=example,dc=com
...
```

Beispieldaten (komplett)

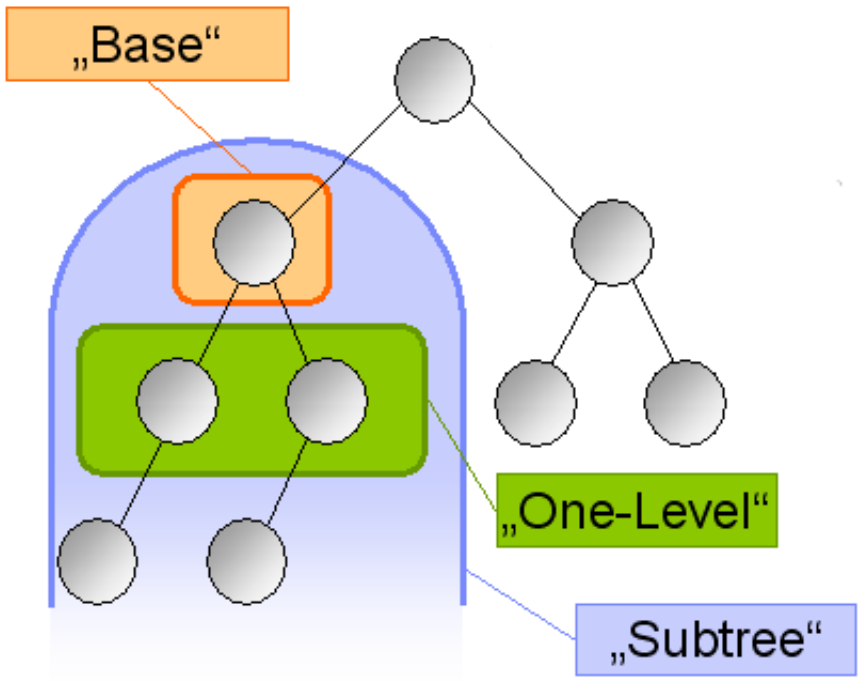


Grundlegende Operationen

Search	Suchen
Add	Hinzufügen neuer Einträge
Delete	Löschen
Modify	Ändern von Attributen
ModifyDN	Verschieben eines Eintrages
Compare	Überprüft ob ein Eintrag einen bestimmten Attributwert enthält
Bind	Anmelden am Verzeichnisserver, ggf mit Authentifizierungsinformationen
Unbind	Abmelden

Suche

- Parameter
  - Startknoten der Suche
  - Umfang (Scope)
    - baseObject
    - singleLevel
    - whole Subtree
  - Suchfilter



Suchfilter

- Typen von Filtern

Filter	Operator	Beispiel
Gleichheit	=	sn=Carter
Substring	=	sn=C*
Ähnlichkeit	=~	sn=~Cartem
Größer/Kleiner	>=, <=	createTimestamp<=20080526142306Z
Existenz	=*	mail=*

- Boolsche Operatoren
  - NICHT: `(!(filter))`
  - UND: `(&(filter1)(filter2)(filter3))`
  - ODER: `(|(filter1)(filter2)(filter3))`

Suchfilter Beispiel

- Finde alle Mitarbeiter des Standorts Santa Clara, für die keine Telefonnummer hinterlegt ist
  - `(&(l=Santa Clara)(!(telephoneNumber=*)))`
- Finde alle Gruppen in denen Mike Carter Mitglied ist
  - `(uniqueMember=uid=mcarter,ou=People,dc=example,dc=com)`

Authentifizierung

- Vor dem Ausführen einer Operation muss sich ein Benutzer am Server anmelden
- Authentifizierungsmechansimen
  - Anonyme Authentifizierung (Anonymous Bind)
  - Passwortbasiert Authentifizierung (Simple Bind)
  - Simple Authentication and Security Layer (SASL)
    - Digest MD5, CRAM MD5
    - GSSAPI (Kerberos)
    - Zertifikatsbasierte Authentifizierung

Eigenschaften von LDAP zusammengefasst

- Optimiert für
  - Schnellen Verbindungsauf- und abbau
  - einfach Suchoperationen und Lesezugriffe
  - verteilte Datenhaltung
    - Replikation
    - Partitionierung
- LDAP-Anfragen
  - verglichen mit SQL deutlich eingeschränkter Funktionsumfang
    - keine Sortierung (ORDER BY), keine arithmetischen Funktionen (COUNT, SUM), DISTINCT
    - keine referentielle Integrität

JNDI (Java Naming & Directory Interface)

Java-APIs zum Zugriff auf LDAP

- JNDI
  - Momentane Standard-API
  - In Java SE enthalten
  - Packages
    - javax.naming.directory (für Standard LDAP Operationen)
    - javax.naming.ldap (für spezielle LDAP Operationen)
- Weitere APIs
  - Novell LDAP Classes for Java
    - Mittlerweile als OpenSource JLDAP bei OpenLDAP
  - Netscape Directory SDK for Java
    - Mittlerweile bei Mozilla
    - Seit längerem keine aktive Weiterentwicklung

JNDI Begriffsklärung

- LDAP abstrahiert (javax.naming)
  - Klassen: Context, DirContext, Name
  - Methoden: bind(), unbind(), createSubcontext(), destroySubcontext(), getNameInNamespace()
- LDAP konkreter (javax.naming.directory)
  - Klassen: Attributes, Attribute, SearchResult, LdapName, Rdn
  - Methoden: search, modifyAttributes

Verbindung zum LDAP Server

- Keine direkte Einflussnahme über Aufbau und Abbau der LDAP-Verbindung, SPI-spezifisch
- Context / DirContext / LdapContext
  - Zentraler Einstiegspunkt für LDAP-Operationen
- Verbindungsparameter
    - Host, Port, Verschlüsselung
    - Authentifizierung (anonymous, simple, SASL)
    - Connection Pooling
  - Programmatisch: new InitialDirContext (Hashtable)
  - Deklarativ: jndi.properties im Classpath

```
DirContext ctx = new InitialDirContext();
ctx.close();
```

Einträge lesen / suchen

- DirContext.getAttributes(Name, String[])
  - liefert alle oder ausgewählte Attribute mit zugehörigen Werten eines Eintrags zurück
- DirContext.search()
  - mächtige Suchmethode bei der Arbeit mit LDAP
    - Suchbasis
    - benutzerdefinierte Filter
    - Suchtiefe
    - zurückgelieferte Attribute
  - liefert NamingEnumeration<SearchResult> zurück
    - getNameInNamespace()
    - getAttributes()

Einträge anlegen

- DirContext.createSubcontext(Name, Attributes)
  - Nicht bind(), würde Attribute serialisieren!
  - Eltern-Eintrag muss existieren

```
Attributes attributes = new BasicAttributes();
Attribute objectClassAttr = new
    BasicAttribute( "objectClass" );
objectClassAttr.add( "top" );
objectClassAttr.add( "person" );
attributes.put( objectClassAttr );
attributes.put( "cn", "Jim Smith" );
ctx.createSubcontext(
    "uid=jsmith,ou=people,dc=example,dc=com",
    attributes );
```

Einträge löschen

- DirContext.destroySubcontext(Name)
  - Funktioniert eigentlich nur mit Blatt-Einträgen
    - Rekursives suchen/löschen
    - Einige LDAP-Server erlauben trotzdem das Löschen von Teilbäumen

```
ctx.destroySubcontext(
    "uid=jsmith,ou=people,dc=example,dc=com" );
```

Einträge ändern

- DirContext.modifyAttributes(Name, ModificationItem[])
  - Gezieltes ändern von Attributen
    - DirContext.ADD_ATTRIBUTE:
      - Hinzufügen eines Attributes oder Wert(e) zu einem bestehenden Attr.
    - DirContext.REMOVE_ATTRIBUTE:
      - Löschen eines Attributes oder einzelner Werte
    - DirContext.REPLACE_ATTRIBUTE:
      - Ersetzen eines Attributes inkl. aller Werte

```
ModificationItem[] mods = new ModificationItem[1];
mods[0] = new ModificationItem(
    DirContext.ADD_ATTRIBUTE,
    new BasicAttribute( "roomNumber", "0123" ) );
ctx.modifyAttributes(
    "uid=jsmith,ou=people,dc=example,dc=com",
    mods );
```

Fazit JNDI

- JNDI ist nicht die optimale API für LDAP
- Einarbeitungsaufwand
- Viel Boilerplate Code
  - try-catch-finally Blöcke
  - NamingException
- Neue Sprachfeatures sind nicht eingezogen
  - Keine for-each-loop wegen NamingEnumeration
  - Keine vargs bei Attributes.put()

Object-LDAP Mapping

Übersicht Objekt-LDAP Mapping

- Grundidee: Objekt-Relationales Mapping
  - JDO, JPA, Hibernate
- Herausforderungen
  - Objekt wird zum Eintrag
  - Mappen von Datentypen
    - einfache Datentypen
    - Arrays, Collections, Maps
    - Objektreferenzen
  - Keine referentielle Integrität
  - Keine Transaktionen
  - Schema dynamisch erweitern
- Beispieldaten: Person und Group, N:M

JNDI Serialisierung

- Java-Objekt wird als byte[] in LDAP abgelegt
  - Klasse muss Serializable implementieren
- Nutzung der Methoden
  - DirContext.bind(Name, Object)
  - DirContext.lookup(Name)
- Vorteil
  - Einfach zu nutzen
- Nachteile
  - Nur durch Java-Applikationen nutzbar
  - Keine LDAP-Suche möglich
  - Nicht für existierenden Datenbestand nutzbar

JNDI State Factories

- Java-Objekt wird beim Speichern in LDAP-Eintrag übersetzt
- Funkionsweise
  - Implementieren von DirStateFactory
  - Factory über java.naming.factory.state registrieren
  - Speichern mit DirContext.bind(Name, Object)
- Nachteile
  - Update mit rebind() macht lookup() + unbind() + bind()
  - Keine Unterstützung für Referenzen
  - Mapping erfolgt programmatisch

JNDI Object Factories

- LDAP-Eintrag wird beim Lesen in Java-Objekt übersetzt
- Funkionsweise
  - Implementieren von DirObjectFactory
  - Factory über java.naming.factory.object registrieren
  - Lesen mit
    - DirContext.lookup(Name)
    - DirContext.search() + setzen von SearchControls.setReturningObjFlag(true)
- Nachteil
  - Keine Unterstützung für Referenzen
  - Mapping erfolgt programmatisch

Spring-LDAP

- Vereinfachung zu JNDI
- LdapTemplate ist zentrale Zugriffsklasse
- Ähnliches Konzept wie State/Object Factory
  - Lesen: AttributeMapper, ContextMapper
  - Schreiben: DirContextAdapter, DirContextOperations
- Unterstützung für Update Operationen
- Nachteil
  - Keine Unterstützung für Referenzen
  - Mapping erfolgt programmatisch

JPOX/Datanucleus

- Implementierung von JDO 1 und 2 sowie JPA
- Datenspeicher: RDBMS, db4o, XML, LDAP
- Vorteile
  - Wenig Einarbeitung, wenn JDO Know-How vorhanden
  - Mapping durch Konfiguration (XML oder Annotations)
  - Referenzen werden unterstützt
- Nachteil
  - LDAP Funktionalität noch in der Entwicklung

Fazit Objekt-LDAP Mapping

- JNDI
  - Object Factories zum Lesen von einfachen Objekten
  - State Factories lieber nicht
- Spring LDAP
  - Vereinfachung zu JNDI
- JPox/Datanucleus
  - Interesanter Ansatz, aber noch nicht für den produktiven Einsatz geeignet
- Keine optimale Lösung vorhanden
  - Eigenimplementierung (DAO + JNDI)

LDAP-Server „embedded“

Übersicht LDAP „embedded“

- LDAP Server aus einer bestehenden Java-Anwendung starten
- Analog zu DerbyDB oder HSQLDB
- Einsatzgebiete
  - Datenspeicher für Benutzer und Rechte
    - Application Server haben einen embedded LDAP Server (BEA WebLogic, Apache Geronimo)
  - Unit-Testen von LDAP Schnittstellen
    - Apache Geronimo nutzt ApacheDS
    - JBoss nutzt OpenDS

Unittests mit ApacheDS

- JUnit3
  - Basisklasse AbstractServerTest
  - In der JUnit setUp() Methode wird der LDAP Server gestartet und die Testdaten initialisiert
  - In den test...() Methoden steht dann ein DirContext zur Verfügung
- JUnit4
  - @RunWith(CiRunner.class)
  - LDAP Server wird einmalig beim Start der JUnit Tests gestartet
  - Änderungen der Daten während eines Tests können nach dem Test zurückgerollt werden
  - Work in progress

Unittests mit ApacheDS

- Beispiel: Testen der State Factory und Object Factory

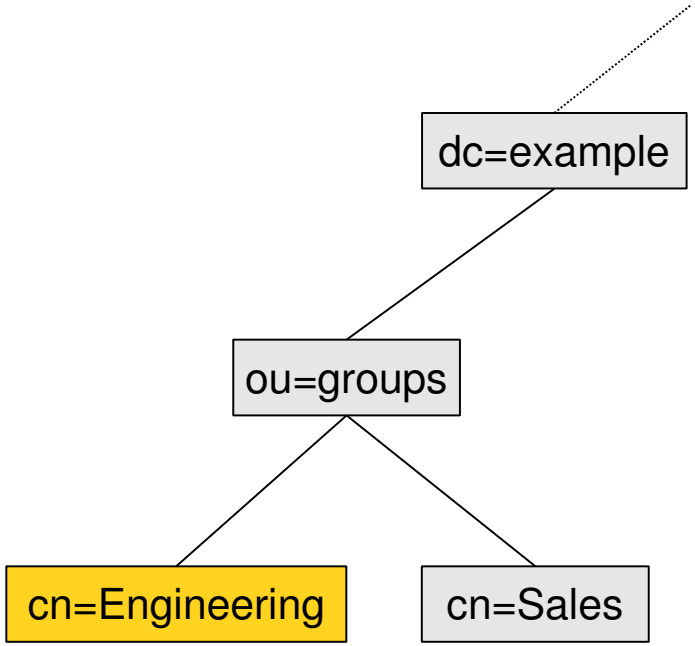
LDAP & Web-Applikationsserver

Sicherheit in der Java EE

- Laufzeitumgebung stellt Basisfunktionen bereit
  - Authentifizierung
  - rollenbasierte Zugriffskontrolle
- Deklarative Sicherheit
  - Konfiguration über Deployment Descriptor web.xml
  - Deklaration von
    - zugriffsgeschütztem Bereich (URL-Pattern)
    - Authentifizierungsmechanismus
    - Rollen

Beispiel Szenario

- Nur die Mitglieder der Gruppe Engineering sollen auf den geschützten Bereich einer Webanwendung zugreifen dürfen



Beispiel: Deployment Descriptor

```
<web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/protected/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
 </web-resource-collection>
 <auth-constraint>
    <role-name>Engineering</role-name>
 </auth-constraint>
```

```
<auth-method>FORM</auth-method>
<realm-name>Example Form-Based Authentication Area</realm-name>
<form-login-config>
   <form-login-page>/login.jsp</form-login-page>
   <form-error-page>/error.jsp</form-error-page>
</form-login-config>
```

```
  <security-role>
    <role-name>Engineering</role-name>
    <role-name>Sales</role-name>
  </security-role>
```

Sicherheit in der Java EE (2)

- Programmatische Sicherheit
  - Ergänzung zur deklarativen Sicherheit
  - Nutzung der beim Login-Vorgang abgefragten Informationen
    - Methoden des Interfaces HttpServletRequest
    - Benutzer-Objekt: `java.security.Principal getUserPrincipal()`
    - Benutzername: `String getRemoteUser()`
    - Rolle: `boolean isUserInRole(String role)`

JNDI Realm für Tomcat

- Realms in Tomcat
  - Definieren, wo Informationen über Benutzer und Rollen gespeichert sind
- Kann über die server.xml konfiguriert werden
  - für bestimmte Anwendungen oder
  - den gesamten Server
- Funktionsweise
  - Verbindungsaufbau zum Server
  - Identifikation des Benutzers
  - Authentifizierung
  - Ermittlung der Rollen

Beispiel: Konfiguration des JNDI Realm

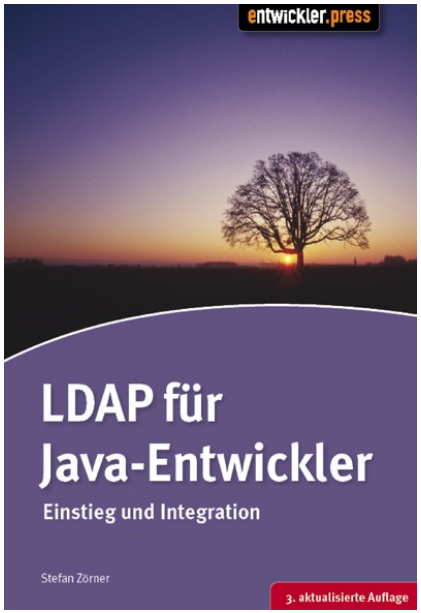
```
<Realm className="org.apache.catalina.realm.JNDIRealm"
    debug="99"
   connectionURL="ldap://localhost:10389"
    connectionName="uid=admin,ou=system"
    connectionPassword="secret"

    userPattern="uid={0},ou=people,dc=example,dc=com"

   roleBase="ou=groups,dc=example,dc=com"
   roleName="cn"
   roleSearch="(uniqueMember={0})"
/>
```

Beschränkungen der Basisfunktionen

- Nutzer-Registrierung
- Passwort-Änderung/Passwort-Policies
- Anzahl fehlgeschlagener Login-Versuche
- sichere Datenübertragung

Weitere Informationen

- LDAP für Java-Entwickler,
  Stefan Zörner, Entwickler.Press, 2007



- Apache Directory Projekt
  http://directory.apache.org
- The JNDI Tutorial (umfangreich, Java 1.4):
  http://java.sun.com/products/jndi/tutorial/
- Trail JNDI (kürzer, Java 5 + 6):
  http://java.sun.com/docs/books/tutorial/jndi/