Bundesverwaltungsamt

IsyFact-Standard

Nutzungskonzept Spring-LDAP

Version 1.7
27.03.2015

IsyFact Standards

Ansprechpartner:

Referat Z II 2
Bundesverwaltungsamt
E-Mail: isyfact@bva.bund.de
Internet: www.isyfact.de

Dokumentinformationen

Dokumenten-ID:	Nutzungskonzept_Spring_LDAP.docx

Inhaltsverzeichnis

1. Einleitung

Im Rahmen der IsyFact wird eine einheitliche Implementierung verwendet, um via LDAP auf Verzeichnisdienste zuzugreifen, in denen Daten zu den verschiedenen Nutzern der Anwendungen einer Plattform gespeichert werden. In diesem Dokument wird der Einsatz von Spring LDAP exemplarisch beschrieben.

Spring LDAP ist eine Erweiterung des Frameworks Spring, das innerhalb der IsyFact-Standards verwendet wird [AnwendungskernDetailkonzept].

Dieses Dokument richtet sich an Architekten und Entwickler, die IsyFact-konforme Anwendungen entwickeln möchten.

2. Funktionalität und Fähigkeiten von Spring LDAP

Hauptziel von Spring LDAP ist das Vereinfachen von Zugriffen auf LDAP-Systeme, wobei folgende wiederkehrende Tätigkeiten gekapselt werden:

- Fehlerbehandlung, alle LDAP-Exceptions werden in Runtime Exceptions gekapselt
- Durchlauf von Attributen mit mehreren Werten
- Suchen und laden von Daten im LDAP
- Verbindungen zum LDAP aufbauen und schließen

3. Nutzung von Spring LDAP

3.1. Benötigte Bibliotheken

Alle von Spring LDAP benötigten Bibliotheken können zusammen mit den eigentlichen Spring LDAP Bibliotheken von [SpringLDAP] herunter geladen werden.

3.2. Spring Konfiguration

In der Spring-Konfigurationsdatei müssen drei Einträge für die Nutzung von Spring LDAP gesetzt werden:

```
<bean id="contextSource"
      class="org.springframework.ldap.pool.factory.PoolingContextSource">
  <property name="contextSource">
    <bean class="org.springframework.ldap.core.support.LdapContextSource">
      <property name="url" value="${ldap.url}" />
      <property name="userDn" value="${ldap.userdn}" />
      <property name="password" value="${ldap.password}" />
      <property name="base" value="${ldap.basedn}" />
      <property name="pooled" value="false" />
    </bean>
  </property>
  <property name="dirContextValidator">
    <bean
class="org.springframework.ldap.pool.validation.DefaultDirContextValidator" />
  </property>
  <property name="maxActive" value="${ldap.maxActive}" />
  <property name="maxTotal" value="${ldap.maxTotal}" />
  <property name="maxIdle" value="${ldap.maxIdle}" />
  <property name="minIdle" value="${ldap.minIdle}" />
  <property name="maxWait" value="${ldap.maxWait}" />
  <property name="whenExhaustedAction" value="${ldap.whenExhaustedAction}" />
  <property name="testOnReturn" value="${ldap.testOnReturn}" />
  <property name="testOnBorrow" value="${ldap.testOnBorrow}" />
  <property name="testWhileIdle" value="${ldap.testWhileIdle}" />
  <property name="timeBetweenEvictionRunsMillis"
            value="${ldap.timeBetweenEvictionRunsMillis}" />
  <property name="numTestsPerEvictionRun"
            value="${ldap.numTestsPerEvictionRun}" />
  <property name="minEvictableIdleTimeMillis"
            value="${ldap.minEvictableIdleTimeMillis}" />
</bean>

<bean id="ldapTemplate"
      class="org.springframework.ldap.core.LdapTemplate">
  <constructor-arg ref="contextSource" />
</bean>

<bean id="ldapTemplateHolder"
      class="bva.bund.de.testdurchstich.springldap.LdapTemplateHolder">
  <property name="ldapTemplate" ref="ldapTemplate" />
</bean>
```

In der Bean vom Typ „`LdapContextSource`“ werden die zum Zugriff auf den LDAP benötigten Parameter definiert. Diese Bean wird so konfiguriert, dass sie kein Pooling durchführt (`pooled = false`). Andernfalls würde der LDAP-Pool des JDKs verwendet, welcher keine Prüfung von Verbindungen erlaubt und somit nach einem Failover des LDAPs defekte Verbindungen im Pool behält.

Anstelle des JDK-Pools wird die Implementierung von Spring verwendet. Dazu wird die `LdapContextSource`-Bean durch eine `PoolingContext-Source`-Bean gekapselt. Letztere führt das Pooling der LDAP-Verbindungen durch. In dieser Bean wird folglich auch der Pool konfiguriert, insbesondere das Prüfen der Verbindungen vor deren Verwendung (`testOnBorrow = true`).

Die Bean „`ldapTemplate`“ definiert die Klasse, die den Zugriff auf den LDAP kapselt, sie benötigt nur die Bean „`contextSource`“ als Parameter.

Da gemäß [AnwendungskernDetailkonzept] Abschnitt 4.1.5 keine Beans für DAO-Objekte definiert werden dürfen, muss die Bean `LdapTemplate` in eine statische Klasse gesetzt werden, den `LdapTemplateHolder`. Dieser bietet das `LdapTemplate` dann über statische Methoden den jeweiligen DAO-Klassen an.

Zum Setzen der Properties der LDAP-Verbindung bzw. des Pools kann folgende Vorlage verwendet werden:

```
# Parameter für die Verbindung zum LDAP
ldap.url = ldap://data.local.vm:1392
ldap.userdn = cn=Directory Manager
ldap.password = DirectoryManager
ldap.basedn = c=de

# Konfiguration des Verbindungs-Pools der Komponente
# Maximale Anzahl von aktiven LDAP-Verbindungen.
ldap.maxActive = -1
# Maximale Anzahl von LDAP-Verbindungen insgesamt (aktiv oder idle)
# (-1 = unendlich).
ldap.maxTotal = -1
# Maximale Anzahl von nicht benutzen Verbindungen im Pool.
ldap.maxIdle = 8
# Minimale Anzahl von nicht benutzen Verbindungen im Pool.
ldap.minIdle = 0
# Maximale Zeit, die auf eine Verbindung gewartet werden soll,
# wenn die maximale Anzahl erreicht ist (-1 = unendlich).
ldap.maxWait = -1
# Bestimmt das Verhalten wenn die maximale Anzahl von Verbindungen erreicht ist
# (0 = Fehler, 1 = Warten, 2 = Pool vergrößern).
ldap.whenExhaustedAction = 2
# Verbindungen werden validiert, wenn sie in den Pool zurückgegeben werden.
ldap.testOnReturn = false
# Verbindungen werden validiert, bevor sie verwendet werden.
ldap.testOnBorrow = true
# Verbindungen im Pool werden in regelmäßigen validiert (EvictionRun)
ldap.testWhileIdle = false
# Zeit zwischen den 'EvictionRuns'.
ldap.timeBetweenEvictionRunsMillis = -1
# Anzahl der Verbindungen, die in einem 'EvictionRun' geprüft werden.
ldap.numTestsPerEvictionRun = 3
# Anzahl der Millisekunden, bevor eine Verbindung für das Herauswerfen
# in einem EvictionRun in Frage kommt.
ldap.minEvictableIdleTimeMillis = 1800000
```

Um gekappte Verbindungen zeitnah zu entfernen und nicht erst, wenn sie benötigt werden, können die folgenden Parameter aus der obigen Beispielkonfiguration wie folgt gesetzt werden:

```
# Zeit zwischen den 'EvictionRuns'.
ldap.timeBetweenEvictionRunsMillis = 5000
# Anzahl der Verbindungen, die in einem 'EvictionRun' geprüft werden.
ldap.numTestsPerEvictionRun = 8
# Anzahl der Millisekunden, bevor eine Verbindung für das Herauswerfen
# in einem EvictionRun in Frage kommt.
ldap.minEvictableIdleTimeMillis = 150000
```

3.3. Beispiel für eine DAO-Klasse

Der hier gezeigte Code dient zum Auslesen der Rollen eines Benutzers sowie zum Anlegen eines neuen Anwenders und ist zentraler Teil der Beispielimplementierung für Spring LDAP. Es wird exemplarisch gezeigt, wie über das `LdapTemplate` Suchen und Einfügen in den LDAP funktioniert.

3.3.1 Auslesen von Rollen

```
public List<String> getRollen(String uid, String orgknz) {
    AndFilter filter = new AndFilter();
    filter.and(new EqualsFilter("uid", uid));
    filter.and(new EqualsFilter("orgknz", orgknz));

    List alleTreffer =
            LdapTemplateHolder.getLdapTemplate().
                 search(DistinguishedName.EMPTY_PATH, filter.encode(),
                 new RollenContextMapper());
    if (alleTreffer == null || alleTreffer.size() == 0) {
        throw new RuntimeException("Kein Benutzer gefunden");
    }
    return (List<String>)alleTreffer.get(0);
}

private static class RollenContextMapper extends AbstractContextMapper {
    public Object doMapFromContext(DirContextOperations ctx) {
        List<String> ergebnis = new ArrayList<String>();
        String[] rollen = ctx.getStringAttributes("rollen");
        for (String rolle : rollen) {
            ergebnis.add(rolle);
        }
        return ergebnis;
    }
}
```

Aufgerufen wird in diesem Beispiel die obere Methode mit `uid` (User-ID) und `orgknz` (Organisationskennzeichen) eines Anwenders, womit ein Anwender eindeutig identifiziert ist.

In den ersten drei Zeilen wird die Suchbedingung definiert, wobei „`uid`“ und „`orgknz`“ die Namen der Entsprechenden Felder im LDAP sind.

In dem Block dahinter wird über den `LdapTemplateHolder` das `LdapTemplate` geholt, und auf diesem die Methode `search` aufgerufen. Dieser Methode wird zuerst ein einschränkender Pfad übergeben, dann die Suchbedingung und danach die Abbildungsregel für das Ergebnis. Als einschränkender Pfad wird eine Konstante für den leeren Pfad übergeben, die Suchbedingung haben wir definiert und als Abbildungsregel wird eine neue Instanz von `RollenContextMapper`

verwendet. Das Ergebnis der Suche wird dann zurückgegeben. Falls es zu keinem Treffer gekommen ist wird eine Exception geworfen.

Die Klasse `RollenContextMapper` definiert das Abbilden von LDAP-Attributen auf Java-Objekte. Die Methode `doMapFromContext` wird einmal für jeden gefundenen Treffer aufgerufen, der übergebene Context enthält alle Werte des Treffers und zusätzliche Metainformationen. In unserer Klasse werden alle Rollen (Inhalt des LDAP-Attributes „`rollen`“) des Benutzers ausgelesen und als Liste zurückgegeben.

Zusammengefasst sucht diese Methode einen Benutzer der durch seinen Anmeldenamen und sein Organisationskennzeichen identifiziert wird, und gibt die Rollen des Benutzers als Liste von Strings zurück.

3.3.2 Speichern eines Anwenders

Als Beispiel zum Speichern wird hier das Neuanlegen eines Anwenders gezeigt. Die Klasse Anwender ist ein reines Transportobjekt mit Getter- und Setter- Methoden und wird nicht weiter erläutert.

```
    public void speicherAnwender(Anwender anwender) {
        Name dn = buildDn(anwender);
        DirContextAdapter adapter = new DirContextAdapter(dn);
        adapter.setAttributeValues("objectclass", new String[] {"top",
                    "person", "organizationalperson", "anwender"});
        adapter.setAttributeValue("cn", anwender.getBenutzerName());
        adapter.setAttributeValue("sn", anwender.getNachName());
        adapter.setAttributeValue("orgknz", anwender.getOrgknz());
        adapter.setAttributeValues("rollen", anwender.getRollen());
        adapter.setAttributeValue("uid", anwender.getUid());
        adapter.setAttributeValue("passwort", "InitialPasswort");
        adapter.setAttributeValue("status", "gueltig");
        LdapTemplateHolder.getLdapTemplate().bind(dn, adapter, null);
    }

    private Name buildDn(Anwender anwender) {
        DistinguishedName name = new DistinguishedName();
        name.add("o", anwender.getOrganisation());
        name.add("ou", anwender.getBehoerde());
        name.add("cn", anwender.getBenutzerName());
        return name;
    }
```

In der ersten Zeile der Methode wird die Methode `buildDn` aufgerufen die den Distinguished-Name des Objektes zusammenbaut. Der Distinguished-Name dient zur eindeutigen Identifizierung eines Anwenders, sein Aufbau ist vom Schema des LDAP abhängig.

In den weiteren Zeilen wird ein Context-Adapter mit den Werten des Anwenders befüllt, wobei jeweils angegeben werden muss, welches LDAP-Attribut mit welchem Wert befüllt wird. Bei der Befüllung muss darauf geachtet werden, dass alle Pflichtattribute der angegebenen Objektklassen gesetzt werden, das Attribut „`objectclass`“ ist immer Pflicht.

In der letzten Zeile der Methode wird wiederum das `LdapTemplate` aufgerufen und mit der Methode bind ein neuer Eintrag im LDAP angelegt. Als erster Parameter wird der DN des Eintrags mitgeliefert, in den

Parametern zwei und drei werden alle zu setzenden Attribute übergeben, entweder als Context oder als Sammlung von Attributen.

4. Quellenverzeichnis

[AnwendungskernDetailkonzept]

Detailkonzept der Komponente Anwendungskern
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Anwendungskern.pdf .

[SpringLDAP]

Spring LDAP
http://www.springframework.org/ldap. (Zugriff am 10.12.2014).