Shib2IdpConnectionPool

Mi is az a connection pooling?

A Java webalkalmazások általában többszálú, hosszú életciklusú környezetben futnak, az adatbázis kapcsolatok azonban természetükből fakadóan nem szálbiztosak (egy kapcsolaton egyszerre csak szál dolgozhat). Ezért szükség van arra, hogy a feldolgozó szálakhoz adatbázis kapcsolatokat rendeljünk. Megjegyzendő, hogy ez nem újdonság, ugyanis PHP esetén például minden egyes futáskor új kapcsolat nyílik.

A kapcsolatkezelésre tehát alapvetően három módszer ismert:

- egy adatbáziskapcsolat, szinkronizációval. Ebben az esetben az alkalmazásunk tulajdonképpen egyszálúvá válik.
- feldolgozó szálanként külön adatbáziskapcsolat, így nem kell a szinkronizációval foglalkozni. Így azonban a kapcsolatok jelentős része kihasználatlan, szélsőséges esetben az adatbázis által engedélyezett kapcsolatok száma telítődhet is. Ez a megoldást tehát jelentős overheadet okoz.
- az adatbázis kapcsolatok dinamikus kiosztása a feldolgozó szálak között.

A fenti három megoldás közül az első kettő webes környezetben teljesítmény okokból erősen ellenjavallt, csak a harmadik módszer a járható út: az adatbázis kapcsolatokat tehát érdemes "készletezni" (connection pooling), és a kapcsolatot kérő szálakhoz futás közben, igény szerint hozzárendelni őket.

Ezen paradigma hatékony működéséhez rendkívül fontos, hogy az alkalmazás csak annyi ideig használja a kapcsolatot, ameddig szükséges, majd azonnal jelezze a pool felé, hogy a kapcsolat szabad (ezt tulajdonképpen a kapcsolat API-szintű lezárásával jelzi, de ilyenkor a fizikai kapcsolat természetesen nem bomlik fel, az másik szál számára ismét kiajánlhatóvá válik).

Az elterjedtebb connection pool implementációk rengeteg segítséget képesek nyújtani:

- minimális és maximális kapcsolatszám meghatározása
- inaktivitás esetén lezárás
- adatbázis oldali lezárás megakadályozása folyamatos "pingeléssel"
- halott kapcsolatok transzparens eltávolítása

A Java alkalmazásszervereknek egy szabványos módszert kell biztosítaniuk az adatbázis kapcsolatok elérésére. Az alkalmazások egy ún. JNDI erőforráson keresztül képesek elérni a kapcsolatok menedzseléséért felelős osztályt (ami általában egy ún. DataSource interfészt implementál). Fontos megemlíteni, hogy ilyenkor az adatbáziskapcsolat felépítését az alkalmazásszerver végzi, így az elérés paramétereit (url, felhasználónév, jelszó) ott kell adminisztrálni, és nem az alkalmazás saját konfigurációjában. Ez lehetőséget ad az adminisztrátor számára, hogy az egyes alkalmazásspecifikus beállítások megtanulása nélkül képes legyen egyik adatbázisról a másikra migrálni.

"Hivatalos", Sun-féle leírás

Connection pool használata Tomcat6 alatt

A Tomcat jelenleg a dbcp nevű pool implementációt használja, ami elég régi és a teljesítménye sem a legjobb, de legalább működik.

Az alábbi példakód egy MySQL kapcsolatot állít be (`/etc/tomcat6/server.xml`), ami kapcsolat-ellenőrzést is végez, mielőtt az alkalmazásnak válaszolna:

```
  <GlobalNamingResources>

    <!-- Create connection pool for idptest.
         Connections will be validated before handed over to the application,
         and every 10 minutes. -->
    <Resource name="jdbc/mymysql" auth="Container"
              type="javax.sql.DataSource"
              driverClassName="com.mysql.jdbc.Driver"
              maxActive="10" maxIdle="2" maxWait="10000"
              username="username" password="password"
              url="jdbc:mysql://localhost:3306/database"
              testWhileIdle="true" timeBetweenEvictionRunsMillis="6000000"
              testOnBorrow="true" validationQuery="SELECT 1" />
  </GlobalNamingResources>
```

A fenti globális JDNI erőforrást egyes alkalmazásokhoz kell rendelni a Context leíróban (pl `/etc/tomcat6/Catalina/localhost/idp.xml`):

```
<Context docBase="...." ... >
   <ResourceLink name="jdbc/mymysql"
                 global="jdbc/mymysql"
                 type="javax.sql.DataSource" />
</Context>
```

A fenti konfiguráció elkészülte után az alkalmazás a `java:comp/env/jdbc/mymysql` JNDI néven éri el az adatbázis kapcsolatot, valahogy így (hibakezelés nélkül, természetesen):

```
// initialize jndi datasource
Context ctx = new InitialContext();
DataSource dataSource = (DataSource) ctx.lookup("java:comp/env/jdbc/mymysql");
// acquire a new connection
Connection conn = dataSource.getConnection();
try {
  // do something
} finally {
  //don't forget to close the connection in the finally block!
  conn.close();
}
```

Egy fontos megjegyzés

Az alkalmazásszerver és az alkalmazás általában két külön osztálybetöltőt (classloader) használ, ezért ebben az esetben nem az alkalmazás mellé kell csomagolni a megfelelő adatbázis drivert, hanem az alkalmazásszerver által látható helyre. Debian Lenny alatt ez például a következőképpen valósítható meg:

```
sudo aptitude install libmysql-java
sudo ln -s /usr/share/java/mysql.jar /usr/share/tomcat6/lib/
```

IdP konfigurálása

Tegyük fel, hogy alkalmazásszerverünk képes a connection poolingra, és a megfelelő DataSource objektumot a JNDI térben rendelkezésre bocsátja `jdbc/idp` néven.

Attribútumok feloldása

Az IdP leggyakrabban attribútum feloldáshoz (pl. perzisztens azonosítókhoz) használ relációs adatbázist, ezért példaként álljon itt egy DataConnector konfiguráció:

```
<resolver:DataConnector xsi:type="StoredId" ...>
  <resolver:Dependency ref="myLDAP" />
  <ContainerManagedConnection resourceName="java:comp/env/jdbc/idp" />
</resolver:DataConnector>
```

Gyakorlatilag az `ApplicationManagedConnection` mondásokat kell `ContainerManagedConnection`-ra cserélni.

JDBC login modul

Bővebben lásd: Shib2IdpAuth#SQL (JDBC) alapon

uApprove

Az alábbi leírás a uApprove legalább 2.3-as verzióihoz íródtak, korábbiak használata nem javasolt.

Ahhoz, hogy a tomcat által kezelt adatbáziskapcsolat használatára rávegyük a uApprove-ot, az alábbiakat kell tennünk.

```
vim conf/uApprove.properties
```

Az adatbázisbeállításoknál kommentezzük ki a default beállításokat és írjuk be a használni kívánt JNDI resource nevét:

```
#--------------------------------------------------------------------#
# Database configuration                                             #
#--------------------------------------------------------------------#

database.resourceName       = java:comp/env/jdbc/mymysql
#database.driver             = com.mysql.jdbc.Driver
#database.url                = jdbc:mysql://localhost:3306/uApprove
#database.username           = uapprove
#database.password           = password

vim conf/uApprove.xml
```

Tegyük inaktívvá az alapértelmezett uApprove.dataSource bean-t, és helyére tegyük az alábbi definíciót

```
<bean id="uApprove.dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" depends-on="shibboleth.LogbackLogging"
    p:jndiName="${database.resourceName}" p:lookupOnStartup="true"
    p:cache="true" p:proxyInterface="javax.sql.DataSource" />
```

Utolsó lépésként másoljuk be a `shibboleth-idp/lib` és a `shibboleth-idp/war/WEB-INF/lib` könyvtárba az alábbi két jart is. (Alapértelmezés szerint az utóbbi útvonal becsomagolva található a shibboleth-idp/war/idp.war fájlban, ezesetben kicsomagolás --> fájlok bemásolása --> visszacsomagolás a követendő út):

- aopalliance-1.0.jar
- spring-aop-3.0.6.RELEASE.jar

Változat #1
document-uploader hozta létre 2025-08-07 12:04:18 CEST
document-uploader frissítette 2025-08-07 12:04:18 CEST