Multi-DataSource pooling with JDBC-Provider

Introduction

Many OSGi bundles require a pool of database connections, and usually the implementing it themselfes.

In larger multi-package projects resp. OSGi container installations, a whole list of bundles require access to one and the same data source.

This not only is a configuration nightmare, it also produces an unnecessary large number of unnecessary large pools.

The new OSGi bundle `org.clazzes.util.jdbc-provider` allows to configure multiple datasources, which are exported as OSGi services implementing the interface `javax.sql.DataSource`.

Configuring JDBC-Provider DataSources

This package uses the configuration PID org.clazzes.jdbc.provider and may be configured with keys using the following patterns:

Key	Description
`datasource.<datasourcename>.url`	JDBC URL
`datasource.<datasourcename>.username`	JDBC User
`datasource.<datasourcename>.password`	JDBC Password
`datasource.<datasourcename>.validationQuery`	Validation query, executed to ensure the application receives a valid connection

For more supported key patterns take a look at theFull list of JDBC Provider Configuration Keys.

Typical JDBC URLs and validation queries can be found in our JDBC Snippets.

Configuration example

Sample configuration file `/etc/apache-karaf/org.clazzes.jdbc.provider.cfg`:

```
datasource.JPTEST.url = jdbc:mysql://localhost/JPTEST
datasource.JPTEST.username = jptest
datasource.JPTEST.password = jptest321
datasource.JPTEST.validationQuery = SELECT 1
datasource.SPECTRUM.url = jdbc:oracle:thin:@10.1.2.3:1521:XE
datasource.SPECTRUM.maxActive = 8
datasource.SPECTRUM.username = FOO
datasource.SPECTRUM.password = bar
datasource.SPECTRUM.validationQuery = select SYSDATE from DUAL
```

Developer Snippets

pom.xml Snippets

It's important for the maven-bundle-plugin to import the Package `javax.sql`:

```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <configuration>
        <instructions>
           ...
           <Import-Package>*,javax.sql</Import-Package>
        </instructions>
      </configuration>
    </plugin>
    ...
  </plugin>
  ...
</build>
```

But that's all you need!

Blueprint Snippets

```
<bp:bean id="configProps" class="org.clazzes.util.osgi.ConfigPropertyAccessor" init-method="createDefaultConfig"
>
  ...
  <bp:property name="defaultValues">
    <bp:map>
      <bp:entry key="datasourceName" value="MYDATASOURCE"/>
      ...
    </bp:map>
  </bp:property>
</bp:bean>
...
<bp:bean id="datasourceName" factory-ref="configProps" factory-method="getProperty">
  <bp:argument value="datasourceName" />
</bp:bean>
...
<bp:bean id="datasourceMap" class="org.clazzes.util.osgi.ServiceMap" init-method="initialize">
  <bp:property name="bundle" ref="blueprintBundle"/>
  <bp:property name="keyProperty" value="datasource.name"/>
  <bp:property name="serviceInterface" value="javax.sql.DataSource"/>
</bp:bean>
<bp:bean id="dataSource" factory-ref="datasourceMap" factory-method="getServiceProxy">
  <bp:argument ref="datasourceName"/>
</bp:bean>
...
<bp:bean id="myDao" ...>
  <bp:property name="dataSource" ref="dataSource" />
</bp:bean>
```

The blueprint xml can and should be kept free from any other JDBC stuff like URLs, driver magic and the like.