Developing with Apache Droids in your spring based project.

Table of contents

1 Develope small specialised components!

Following the UNIX philosophy for automated task processing in java enable us to re-use the maximum code from other projects. As a reminder a pipe in unix starts with an invoking component (which produces a stream) and then chain as much other components that interact on the stream that are needed. The modification of each component will be passed to the next component in the chain.

For example the following command in a unix box will lance a subversion command to check for the status on the local svn checkout (svn st). The next command will filter the files that are not under svn control (grep ?). The next command will modify the stream to create a command to add this files to the repository (awk ...). The last step will cause the invocation of the command by sending it to the shell (sh).

```
svn st | grep ? | awk '{print "svn add "$2}' | sh
```

In droids your are piping/processing your tasks with small specialist components that combined are resolving your task.

2 How can I use Apache Droids in my application?

Once you have created the droids.jar, place it in your classpath and you are ready to develope your own components and robots. Droids is completly Spring driven, if you never have seen Spring at all you need to check out their documentation otherwise you will be lost.

We will develope a **SIMPLE** robot that will crawl a web page and send the data to an Apache Solr server (poor man's approach of Apache Nutch crawling) and will call it `"indexer"`.

The resulting classes can be found in `$DROIDS_HOME/src/example/java` and can be compiled with `ant droids.compile-example`. This target compiles the examples, creates a jar and copy the resulting jar into the lib directory.

3 Extend the default spring configuration.

In your principal spring configuration you need to import the default spring configuration files from droids. A typical file would look something like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
    "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <import resource="classpath:/org/apache/droids/droids-core-factories-context.xml"/>
  <import resource="classpath:/org/apache/droids/droids-core-context.xml"/>
<!-- ... -->
</beans>
```

Then we need to configure the org.apache.droids.helper.factories.DroidFactory bean to add our new droid.

```
<bean id="org.apache.droids.helper.factories.DroidFactory"
  class="org.apache.droids.helper.factories.DroidFactory">
  <property name="map">
    <map>
      <entry key="indexer" value-ref="indexer"/>
      <!-- Referencing as well the Hello crawler
         that is configured by the Droids core -->
      <entry key="hello" value-ref="hello"/>
    </map>
  </property>
</bean>
```

Then we need to configure our `indexer`.

```
<!-- Indexer -->
<bean id="indexer" class="org.apache.droids.examples.IndexerCrawler">
  <property name="core" ref="org.apache.droids.Core"/>
  <property name="queue" ref="org.apache.droids.queue.Simple"/>
  <property name="maxThreads" value="@droids.maxThreads@"/>
  <property name="url" value="@droids.initial.url@"/>
  <property name="updateUrl" value="http://localhost:8983/solr/update"/>
</bean>
```

The variables within @@ are replaced while building the jar with ant. The actual value is defiened either by the properties in your build.properties or the default.properties.

Now we need to wrap up our droid in defining our new handler.

```
<bean id="org.apache.droids.helper.factories.HandlerFactory"
  class="org.apache.droids.helper.factories.HandlerFactory">
  <property name="map">
    <map>
      <entry key="solr" value-ref="org.apache.droids.handle.Solr"/>
    </map>
  </property>
</bean>

<!-- Handler -->
<bean id="org.apache.droids.handle.Solr" class="org.apache.droids.handle.Solr">
  <property name="updateUrl" value="http://localhost:8983/solr/update"/>
</bean>
```