Geomajas GeoTools layer plug-in

Geomajas Developers and Geosparc

Geomajas GeoTools layer plug-in

by Geomajas Developers and Geosparc

1.11.2

Table of Contents

1. Introduction 1
2. Configuration 2
    1. GeoTools layer configuration 2
        1.1. Configuring a Web Feature Service (WFS) layer 2
        1.2. Configuring a database layer (PostGIS, Oracle Spatial,...) 3
    2. Non-typed feature ids 5
    3. Shape in memory layer 6
    4. Transaction configuration 6
3. How-to 7
    1. How to recover from connection problems 7

List of Tables

2.1. GeoToolsLayer configuration 2
2.2. ShapeInMemLayer configuration 6

List of Examples

2.1. GeoTools transaction configuration 6

Chapter 1. Introduction

The GeoTools layer allows you to build a Geomajas vector layer from any GeoTools DataStore object.

The layer is built to be able to recover from connection problems. When a data source is temporary unavailable, you will get exceptions during the downtime, but the connection should recover once the data source is available again.

The GeoTools layer also allows you to connect build a layer from a shape file. You can do this using the ShapeInMemLayer, which reads the entire shape file in memory. This does not allow you to save the changes back to disk at the moment (changes are saved in memory but will be lost when the application is restarted).

Alternatively, you can use a GeoTools data store to access a shape file. This will persist changes back to disk but is not safe for concurrent access (which is the case in Geomajas) and thus should only be used for testing[1]. For proper support of shape files, it is recommended to import the shape file into a database (PostGIS for example has a script for this).

---

[1] GeoTools 2.7 indexes the shape file in a data structure which does not handle concurrency.

Chapter 2. Configuration

1. GeoTools layer configuration

This section handles possible configurations using the GeoTools layer. As GeoTools supports a whole range of data formats, not all of them will be covered here.

Apart from a reference to the layer info, you can set the following parameters:

Table 2.1. GeoToolsLayer configuration

Name	**Description**
url	URL for the shape file. Apart from standard protocols supported by Java, you can also use the "classpath:" protocol (the resource location should not start with a slash) to refer to shape files on the class path.
dbtype	Database type, useful when the data store is a database.
parameters	You can define additional parameters which can be passed to the GeoTools data store. These are name/value pairs which are passed in `org.geomajas.configuration.Parameter` objects.
dataSource	javax.sql.DataSource to be used for the database datastore. This allows to configure a different connection pool and reuse it between layers.
cooldownTimeBetweenInitializationRetries	Period in milliseconds that the system has to wait between attempts to connect to the data source. When the layer fails to access the data store, it is discarded and it will try to rebuild the data store on the next access. However, to prevent hammering the server, it will only try to rebuild the data store when the configured time has passed.

1.1. Configuring a Web Feature Service (WFS) layer

In order to read data from a WFS, a GeoTools layer is required. Currently GeoTools has support for WFS 1.0 and 1.1, and support for WFS-T 1.0. As the GeoTools WFS plug-in is not by default part of the class path, you must add it to the class path manually. First things first, the XML configuration. There are 2 ways of configuring a WFS layer through the GeoTools layer. As Geomajas uses a GeoTools DataStore behind the screens, it is possible to configure this DataStore separately, and than attach it to the GeoTools layer. This would have the advantage of re-using the same GeoTools DataStore over multiple layers. Alternatively you can configure the required parameters directly within the Layer configuration.

In general, the following parameters can be configured:

- *WFSDataStoreFactory:GET_CAPABILITIES_URL*: URL for the getCapabilities document on the server instance.
- *WFSDataStoreFactory:PROTOCOL*: determine which HTTP command use when requesting WFS functionality. Set this value to "true" for POST, "false" for GET or NULL for AUTO.
- *WFSDataStoreFactory:USERNAME*: set the user name which should be used to authenticate the connection. This parameter should not be used without the password parameter.
- *WFSDataStoreFactory:PASSWORD*: set the password which should be used to authenticate the connection. This parameter should not be used without the user name parameter.

- *WFSDataStoreFactory:TIMEOUT*: specify the connection timeout in milliseconds. This parameter has a default value of 3000ms.
- *WFSDataStoreFactory:BUFFER_SIZE*: set the buffer size for the features. This parameter has a default value of 10 features.
- *WFSDataStoreFactory:TRY_GZIP*: indicate whether the data store should use gzip compression to transfer data if the server supports it. Default is true.
- *WFSDataStoreFactory:LENIENT*: indicate whether the data store should do its best to create features from the provided data even if it does not accurately match the schema. Errors will be logged but the parsing will continue if this is true. Default is false.

Note that most of the parameters above are optional. Only the capabilities URL is required.

You can configure a WFS layer like this:

```
<bean name="anotherWfsLayer" class="org.geomajas.layer.geotools.GeoToolsLayer">
   <property name="parameters">
      <list>
         <bean class="org.geomajas.configuration.Parameter">
            <property name="name" value="WFSDataStoreFactory:GET_CAPABILITIES_U
            <property name="value" value="http://www.some-wfs.com/ows?service=W
         </bean>
         <bean class="org.geomajas.configuration.Parameter">
            <property name="name" value="WFSDataStoreFactory:TIMEOUT" />
            <property name="value" value="5000" />
         </bean>
      </list>
   </property>
   <property name="layerInfo" ref="blablaInfo" />
</bean>
```

As said in the beginning of this section, the GeoTools WFS library needs to be added as a dependency in your project. When using Maven, you can add the following dependency:

```
<dependency>
  <groupId>org.geotools</groupId>
  <artifactId>gt-wfs</artifactId>
  <version>${geotools-version}</version>
</dependency>
```

1.2. Configuring a database layer (PostGIS, Oracle Spatial,...)

Warning

> The configurations in this section still need to be double checked! If you find errors, please post it on the forum.

Configuring a GeoTools layer that makes use of a PostGIS database, is again a question of using the correct parameters. There is the choice of configuring the datasource separately or using the built-in datasource of GeoTools. By configuring the datasource separately, it can be used by multiple layers. In the case of a database, this method would almost always be preferable.

If you prefer a custom datasource configuration (we recommend BoneCP but alternatives are possible), the dataSource property should be configured:

- dataSource: bean reference or inner bean of type javax.sql.DataSource

In this case only the *dbtype* and *namespace* parameters should be configured:

- *dbtype*: The type of database: (e.g. "postgis").
- *namespace*: The namespace of the datastore (e.g. "postgis").

If using the built-in datasource of GeoTools, the following extra parameters can be configured:

- *database*: The name of the database.
- *user*: The user name to log in with.
- *passwd*: The user's password.
- *host*: The hostname or IP address of the machine where the database is located.
- *port*: The port where the database runs (default for PostGIS is 5432).

Configuring a database layer using PostGIS can be done like this (with separate datasource):

```
<bean id="postgisDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy
   <property name="driverClass" value="org.postgresql.Driver" />
   <property name="jdbcUrl" value="jdbc:postgresql://localhost:5432/app" />
   <property name="username" value="app"/>
   <property name="password" value="app"/>
   <property name="idleConnectionTestPeriod" value="60"/>
   <property name="idleMaxAge" value="240"/>
   <property name="maxConnectionsPerPartition" value="30"/>
   <property name="minConnectionsPerPartition" value="10"/>
   <property name="partitionCount" value="3"/>
   <property name="acquireIncrement" value="5"/>
   <property name="statementsCacheSize" value="100"/>
   <property name="releaseHelperThreads" value="3"/>
</bean>

<bean name="layerCountries" class="org.geomajas.layer.geotools.GeoToolsLayer">
  <property name="layerInfo" ref="layerCountriesInfo"/>
  <property name="dataSource" ref="postgisDataSource"/>
  <property name="parameters">
   <list>
    <bean class="org.geomajas.configuration.Parameter">
     <property name="name" value="namespace" />
     <property name="value" value="postgis" />
    </bean>
    <bean class="org.geomajas.configuration.Parameter">
     <property name="name" value="dbtype" />
     <property name="value" value="postgis" />
    </bean>
   </list>
  </property>
</bean>
```

Don't forget to also add the necessary GeoTools datastore library (typically gt-jdbc-xxx) to your classpath and the required database driver libraries. Here is an example that adds the postgresql driver and (next generation) GeoTools PostGIS JDBC library to the pom.xml (when using Maven):

```
<dependency>
   <groupId>postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <version>8.1-407.jdbc3</version>
```

```
</dependency>
<dependency>
   <groupId>org.geotools</groupId>
   <artifactId>gt-jdbc-postgis</artifactId>
   <version>${geotools-version}</version>
</dependency>
```

Note

When configuring the general layer information (and attribute information), the value may differ, depending on the kind of database that you use. For example, column names in a PostGIS database all have lower cases, and so the Geomajas attribute configuration should reflect this. If you are using Oracle Spatial on the other hand all column names are upper case, so your configuration should contain upper case for the attribute names.

2. Non-typed feature ids

Geomajas always uses the id returned by the feature model. For geotools layers, this is the same id as returned by the datastore and therefore follows any conventions that geotools is imposing. For the next-generation JDBC datastores (gt-jdbc-xxx), this means that the id is `typed`, i.e. constructed as a combination of the type name (table or view name) and the primary key, and separated with a dot: `mytable.1`, `mytable.2`, etc....

For first-generation JDBC datastores (gt-xxx), which support non-typed fids, a special configuration parameter `useTypedFids` has been added. This parameter defaults to true, but can be set to false if one needs the primary key without prefixes as fid:

```
<bean name="layerPointNonTyped" class="org.geomajas.layer.geotools.GeoToolsLaye
    <property name="layerInfo" ref="layerPointNonTypedInfo" />
    <property name="parameters">
        <list>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="namespace" />
                <property name="value" value="postgis" />
            </bean>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="dbtype" />
                <property name="value" value="postgis" />
            </bean>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="database" />
                <property name="value" value="test" />
            </bean>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="user" />
                <property name="value" value="postgres" />
            </bean>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="passwd" />
                <property name="value" value="postgis" />
            </bean>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="host" />
                <property name="value" value="localhost" />
            </bean>
            <bean class="org.geomajas.configuration.Parameter">
                <property name="name" value="port" />
                <property name="value" value="5432" />
```

```
      </bean>
      <bean class="org.geomajas.configuration.Parameter">
        <property name="name" value="useTypedFids" />
        <property name="value" value="false" />
      </bean>
    </list>
  </property>
</bean>
```

3. Shape in memory layer

This layer is backed by a shape file that is loaded in memory at startup. All layer updates are performed in memory, so this layer is not really useful except for examples.

Table 2.2. ShapeInMemLayer configuration

ShapeInMemLayer configuration	
url	URL for the shape file. Apart from standard protocols supported by Java, you can also use the "classpath:" protocol (the resource location should not start with a slash) to refer to shape files on the class path.

4. Transaction configuration

For GeoTools layers, the following transaction configuration can be used:

Example 2.1. GeoTools transaction configuration

```
<!--
  enable the configuration of transactional behavior based on
  annotations
-->
<tx:annotation-driven proxy-target-class="true" transaction-manager="transa

<!--  only necessary if you have no hibernate or datasource transaction man
<bean id="transactionManager" class="org.geomajas.layer.geotools.GeoToolsTr
</bean>
```

There are 2 entries:

- A tag to enable annotation-based transactional behavior, internally used by Geomajas (and your custom code) to decide which commands need transaction support.
- The platform transaction manager: Spring comes with a set of predefined transaction managers that are tied to JDBC, Hibernate, JTA, etc.... You can use any of these, as long as there is at least one transaction manager bean defined. If you have no need for a transaction manager outside of the geotools layer and there is no transaction manager that fits your geotools layer (e.g. your layer is not JDBC-based), you can should use a dummy transaction manager here to enable transaction synchronization: `GeotoolsTransactionManager`.

Chapter 3. How-to

1. How to recover from connection problems

The GeoTools layer automatically recovers from connection problems. When the layer is created an attempt is made to connect to the data source. If that fails a warning is logged and the connection is re-established on the next attempt to use it. There is a short (configurable) grace period between connection attempts to ensure that the service is not hammered.

However this only works if you use the recommended way of configuring the GeoTools layer, that is using the parameters property. If you use the datastore property then the inability to connect to the data source at layer creation time will result in an exception and failed startup (note that this way of configuring is not part of the API and may become impossible at any time).