Connection pool subsystem upgrade

Motivation:	Have flexible, production ready connection pooling for JDBC data stores
Contact:	Andrea Aime
Tracker:	GEOT-1373
Tagline:	

Description

Current Geotools JDBC data stores use a home grown ConnectionPool class which is showing its limitations quite evidently in production environment.
There is plenty of open source and commercial connection pool implementations available, and we should be able to leverage them instead.

The cornerstone of the change is to make JDBC DataStore use DataSource, a standard java interface for Connection providers. Then again, we also want to be able and leverage special connection pools provided by web containers (JNDI), or optimized for a specific database (OC4J for Oracle, for example).

To allow such flexibility, DataSource should be provided by a pluggable system. A DataSourceFactorySPI, by all means a close clone of DataStoreFactorySPI, will gather the connection pool parameters (they may be many) and build a DataSource. New data store factories for JDBC data stores will then accept DataSource as parameter, whilst the old ones will fall back on using a default implementation, with default parameters (DBCP).

Finally, some datastores will need to unwrap the pooled connections, so a separate SPI will be needed to support this unwrapping process (which, before java 6, is connection pool implementation dependent).

More information and discussion can be found on the associated research page.

Overall Design

The proposed change revolves under a main modification in the JDBC data stores and factories, and two new SPI:

- make it so the JDBC data stores use DataSource instead of the old ConnectionPool class (Eclipse reports 126 references to the ConnectionPool class), and deprecate the old ConnectionPool class (note, this will break users that are calling the JDBC data stores constructors directly)
- modify the old factory classes so that they use a default DBCP based connection pool instead of ConnectionPool
- create new JDBC datastore factory classes accepting a DataSource parameter
- create a new SPI, DataSourceFactorySPI, to make DataSource creation pluggable
- create a second new SPI, Unwrapper, to unwrap pooled connections and return native ones

To better test and develop this approach a new branch has been created, http://svn.geotools.org/geotools/branches/2.4-ds, where all the above steps are being attempted and new insights can be gained.

DataSourceFactorySPI

DataSource generators will be pluggable. Since each DataSource can take lots of parameters to be created, we propose the same design as DataStoreFactorySpi, that is, an SPI with parameters that can be introspected, and a finder to lookup them up given a map of parameters.

Instead of pasting the interfaces here, I'll refer the current prototype implementation in svn.
Here are the main SPI interface and the associated finder:

- DataSourceFactorySPI
- DataSourceFinder

And here are the default DBCP and JNDI implementation, as well as their base abstract class:

- AbstractDataSourceFactorySpi
- DBCPDataSourceFactory
- JDNIDataSourceFactory

The unit test show example usage of both:

- DataSourceFinderTest

Unwrapper

As stated, some `DataStore` implementations (Oracle) need to access the native connection, and some others (PostGis) need native Statement instead, whilst the pooling data sources always return wrappers.
UnWrapper is an interface designed to perform the extraction of native connections and statements. A separate interface from DataSourceFactory is needed, because some DataSource factories are opaque, in that they do not know what kind of `DataSource` they are returning (JNDI is the typical case).

The interface will thus have many implementations to target different DataSource providers. Since there is a strict connection with DataSource, `UnWrapper` lookup will be performed again with the DataSourceFinder.

Here are the interface and the finder:

- UnWrapper
- DataSourceFinder

Here is a sample DBCP implementation:

- DBCPUnWrapper

And here is a sample usage in a unit test:

- UnWrapperTest

Status

This propsal is ready for discussion and vote.

- Andrea Aime +1
- Ian Turton +1
- Justin Deoliveira
- Jody Garnett +1
- Martin Desruisseaux +1
- Simone Giannecchini +1

On the 2.4-ds branch the basic interfaces and implementations are shelled out, and Postgis, Postgis versioned, H2 and Oracle datastores have been migrated to use DataSource (all their tests are still passing)

dynamictasklist: task list macros declared inside wiki-markup macros are not supported

Resources

File	Modified

No files shared here yet.

Tasks

	no progress		done		impeded		lack mandate/funds/time		volunteer needed

A target release is also provided for each milestone.

Milestone 1	2.4-M4	
	Andrea Aime	Prototype out DataSourceFactorySPI
	Andrea Aime	Prototype out Unwrapper
	Andrea Aime	Switch all mainstream JDBC data stores to DataSource and the old factories to DBCP (Postgis, Oracle, MySql)
	Andrea Aime	Switch DB2 to DataSource and the old factories to DBCP
	?	Switch GeoMedia to DataSource and the old factories to DBCP (this may just lead GeoMedia datastore out of the build, it's unmantained afaik)
	Andrea Aime	Create new "advanced" factories that use the DataSource parameter

	Jody Garnett	update use documentation
	Jody Garnett	Outline JDBC Guide
	Andrea Aime	Provide unwrapper implementation documentation for JDBC Guide
	Gabriel Roldán	Example JNDI DataSource setup for Tomcat, JBoss, etc...

API Changes

Old public API users won't notice any change, since the old JDBC factories will be kept in place. If you created a JDBC datastore by setting up a map of parameters, and then used either DataStoreFinder or the JDBC datastore factory directly, you should not have any trouble.

Old users that did access the JDBC factories directly will unfortunately be broken, since the `ConnectionPool` parameter will be replaced by a `DataSource` one. Here is a comparisong between old and new code:

Old code

```
PostgisConnectionFactory factory1 = new PostgisConnectionFactory(host, port,
database);
ConnectionPool pool = factory1.getConnectionPool(user, password);

JDBCDataStoreConfig config = JDBCDataStoreConfig.createWithNameSpaceAndSchemaName(
namespace, schema );
data = new PostgisDataStore(pool, config, PostgisDataStore.OPTIMIZE_SAFE);
```

New code

```
DataSource ds = DataSourceUtil.buildDefaultDataSource("jdbc:postgresql://" + host +
":" + port + "/" + database, "org.postgresql.Driver", user, password, null, false);
JDBCDataStoreConfig config = JDBCDataStoreConfig.createWithNameSpaceAndSchemaName(
namespace, schema );
data = new PostgisDataStore(ds, config, PostgisDataStore.OPTIMIZE_SAFE);
```

New users will be able to setup a custom data source with the following code:

Example.java

```
Map map = new HashMap();
map.put(DBCPDataSourceFactory.DRIVERCLASS.key, "org.h2.Driver");
map.put(DBCPDataSourceFactory.JDBC_URL.key, "jdbc:h2:mem:test_mem");
map.put(DBCPDataSourceFactory.USERNAME.key, "admin");
map.put(DBCPDataSourceFactory.PASSWORD.key, "");
map.put(DBCPDataSourceFactory.MAXACTIVE.key, new Integer(10));
map.put(DBCPDataSourceFactory.MAXIDLE.key, new Integer(0));
DataSource source =  DataSourceFinder.getDataSource(map);

map = new HashMap();
map.put("dataSource", source);
map.put("dbType", "postgis");
DataStore store = DataStoreFinder.getDataStore(map);
```

Documentation Changes

Website:

- Update Upgrade to 2.4 instructions

User Guide:

- 01 Use of DataSource
- Update examples to reflect changes to JDBC data stores
- JDBC Guide - we need to create a section showing how to implement an unwrapper (for those working on a new application container we have not met yet)
- JDBC Guide - we need a section showing how to configure DataSource stuff for Tomcat, JBoss, etc...

Issue Tracker:

- check related issues to see of problems are affected