Catalog Improvements

Description

The existing catalog interface has a description, the following is just for reference.

- Catalog: facility to locate (or search for) geospatial resources. This is just an interface and is often implemented against a remote web service.
- LocalCatalog: manages "live" resources, supports the same "search" functionality as Catalog, but also takes responsibility for storing the actual "connections". The LocalCatalog will manage your DataStores and GridCoverages for you.

If we were forced to drop the word catalog we would be left with:

- ResourceLocator - search for resources
- ResourceManager - manage resources, and any connections or data associated with them.

(I would prefer to stick with Catalog and LocalCatalog)

History

The GeoTools Catalog interfaces has the following history:

- GeoServer Data module defined 2003
- GeoServer Data module "lookup methods" isolated as GeoTools Registry API in 2003, with the addition of "cross datastore" operations.
- GeoTools Registry API used to inform uDig Catalog in 2004
- GeoTools catalog derrived from uDig 1.1 catalog in 2006

The wheel has now turned full circle and we are looking to share a common API for this idea between both GeoServer and uDig. The fact that an implementation will also remove the boiler plate datastore managmenent code from our examples (and developer community) is a big plus.

Community Requirements

The following notes have been collected from email, we will try and reduce them to either acceptance tests or guidelines when choosing between alternatives.

Jesse Eichar

- Service and GeoResource is lazy loading. IE obtaining a reference to them will not block.
- There must be a method to obtain error and warning messages. Such as misconfigurations, sub-optimal configurations or broken connections/missing files
- There must be events when the connection to resources change and when cached values change. The events must provide enough information to accurately determine what has changed
- The burden of a contributor must be minimalized. IE there should not be an unreasonable number of classes that they must implement.
- There must be a way to handle session properties. As a use-case consider what happens when a client is operating on a FeatureStore within a transaction. There must be a way to query the GeoResource for the bounds and have it return the correct bounds considering the transaction.

Jody Garnett

- Clear API for developers, I am okay with some complexity for implementers as long as their is a strong usability benefit.
- Preserve the resource to Style association as a user story

Andrea Aime

- Clean up the current mess that is our config/catalog api
- Have a way for people to plug in their special catalog implementations
- Allow for clustering (aka, store config in a central location, be it a database, or a separate Geoserver publishing its configuration to "slave" Geoservers). Please note I've written *allow*, I mean that the API we choose should not go against a db based API, not that we will make one
- Have Geoserver scale on the thousands of feature types/coverages.

Justin Deolivera

- ?

Chris Holmes

- What is the benefit to users of GeoServer?

Alessio Fabiani

- I want the Ingestion Engine save the new configuration somewhere (Disk or DB) independently from GeoServer, and have the GeoServer catalog always updated.
- Don't load all the resources every time into memory, but ask just for the resoucre I need.
- Cache the resources in order to avoid to request them to the service everytime.