Internationalization (Logging & Resources)

The Knox project uses a unique logging and i18n resource mechanism based on interfaces and dynamic proxies.

Basically take a look at these two examples in the Knox code.

```
gateway-server/src/main/java/org/apache/hadoop/gateway/GatewayMessages.java
gateway-server/src/main/java/org/apache/hadoop/gateway/GatewayResources.java
```

and then at the usages of some of these methods in say

```
gateway-server/src/main/java/org/apache/hadoop/gateway/GatewayServer.java
```

Note that private static dynamic proxies are created for the message and resources interfaces.

```
private static GatewayResources res = ResourcesFactory.get( GatewayResources.class );
private static GatewayMessages log = MessagesFactory.get( GatewayMessages.class );
```

Then you can call the methods anywhere in the class.

```
log.failedToParseCommandLine( e );
```

in this the interface method below is being used.

```
@Message( level = MessageLevel.FATAL, text = "Failed to parse command line: {0}" )
void failedToParseCommandLine( @StackTrace( level = MessageLevel.DEBUG ) ParseException e );
```

This logs a message at level FATAL and create the message from the text and the toString of the param. If the logger level is DEBUG or finer the stack trace is also logged.

Resources are similar.

```
System.out.println( res.gatewayVersionMessage(
    buildProperties.getProperty( "build.version", "unknown" ),
    buildProperties.getProperty( "build.hash", "unknown" ) ) );
```

In this case the following resource method is being used.

```
@Resource( text="Apache Hadoop Gateway {0} ({1})" )
String gatewayVersionMessage( String version, String hash );
```

This will merge the version and hash information into the text and return it.

I haven't done it yet but I'll eventually write and APT plugin to extract the text in the annotation out into resource bundles for translation.
Note: My preference is to have one resource and message class for each module.

If you are interested the code can be found at

```
gateway-i18n/src/main/java/org/apache/hadoop/gateway/i18n/messages/MessagesFactory.java
```

Note: The performance key here is to avoid as much as possible any string concatenation or toString() invocations in the code calling the message and resource methods. Pass all the parts to the message method and use the format text to do the concatenation. The implementation of the dynamic proxy will only create the concatenated message if the message will actually be logged. Under the covers it uses the java.text.MessageFormat class so that is the syntax required.