Enhydra Application Utilities: XMLC Deferred Parsing, Automatic Reloading and Memory Persistence

Vladimir Radisic

Table of Contents

Introduction

XMLC Deferred parsing enables you to change a HTML page in a Web application at runtime (which means without stopping the application). So, if you want a new look on a page, you can change desired HTML page in the application. The application detects the new timestamp on the file, dynamically loads the page, and uses it in the application. Configuring deferred parsing to work is also simple once you understand the details behind it. Deferred parsing comes with XMLC2.2 and it represents replacement for the earlier posibility of XMLC, autorecompilation. For more information about deferred parsing refer to XMLC2.2 documentation.

Automatic Reloading also enables you to change the Web application at runtime (which means without stopping the application). This changes can be seen (by web users) only if the application, or parts of application that had been changed, are compiled or transformed from HTML form into Java classes by XMLC, and then put in the place of old application's classes.

Both Deferred parsing and Automatic Reloading enable changes of the Web application without stop-

ping the application and then restarting it, so that Web users, except when the application is seriously changed, can not notice any change in the Web application. The changes in particular Web page are applied when the page is called by HTTP request for the first time. This two approaches can work together. The difference between these two approaches is that in XMLC Deferred Parsing the changed HTML pages are dinamicaly loaded into the system, during the runtime, as org.w3c.dom.Document/org.w3c.dom.html.HTMLDocument wihtout having to pre-compile them by using XMLC, while before performing the Automatic Reloading, it is necessary to compile the changed HTML files (and/or other changed application classes).

Enhydra's possibility called Memory Persistence is used in order to held application's session objects in memory while particular application is stopped, or during the restart of the application. When application is restarted, it can use its old session data (data stored in session object) as they were left in the time the application had been stopped. Of course, it is assumed that used servlet container wasn't stopped in the meantime.

XMLC Deferred Parsing

To run Enhydra application with XMLC Deferred Parsing, the following preparation actions must be done:

- XMLCFactory must be used to instantiate pages in your Java code, i.e.

```
WelcomeHTML welcome =
  (WelcomeHTML)comms.xmlcFactory.create(WelcomeHTML.class);
```

  The creation of Java objects with DOM representation of HTML pages in Enhydra presentation objects, in the case of the Deferred Parsing and Automatic Reloading must be done only by create method of the xmlcFactory class (like in the example above). The creation of instances with the Java key word 'new' must not be performed. This way of creating instances (with 'new') can be used in applications in which Deferred Parsing and Automatic Reloading are not used.

- Appropriate Deferred Parsing parameter must be set in application configuration file (*<app_name>.conf* or *web.xml* file depending on which way of application parameters reading is chosen). If *<app_name>.conf* is used, the parameter should be set as below:

```
Server.XMLC.DeferredParsing
  = true
```

  while the same parameter, in the case of *web.xml* file, should be defined as 'env-entry' tag, shown below:

```
<env-entry>
  <env-entry-name>Server/XMLC/DeferredParsing</env-entry-name>
  <env-entry-value>true</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
```

  Note that the configuration file that will be used for reading application parameters is defined in corresponding application web.xml file (parameters 'ConfFile' and 'ConfFileClass').

  In the case when an application project has been made by 'appwizard', the two application configuration template files should be edited, in order to define Deferred Parsing via <app_name>.conf file:

```
<app_root>/presentation/resource/appwizard/conf/<app_name>.conf.in
  <app_root>/application/resource/appwizard/conf/<app_name>.conf.in
```

Also, the two web.xml files should be edited in order to set Deferred Parsing parameter, via application web.xml file:

```
<app_root>/presentation/resource/appwizard/WEB-INF/web.xml
  <app_root>/application/resource/appwizard/WEB-INF/web.xml
```

- Before any XMLC parsing, in each subdirectory which contains resources (directories with HTML files), file options.xmlc must be edited with following parameter:

```
-for-deferred-parsing
```

This parameter suggests, to later invoked XMLC parser, that resources (HTML files) should be parsed into DOM objects for Deferred Parsing usage.

- After all preparation acts mentioned in the previous tips were done, the building of application project prepared for Deferred Parsing can be done.

In the case when an application project has been made by 'appwizard', the invoking of command on the command line:

```
enhydra-ant
```

will start ant based build process, which has as one of its tasks XMLC parsing of the resources. The results of XMLC parsing are located in directory:

```
<app_root>/presentation/src-generated
```

and they consist of Java DOM class representation for each HTML resource file and corresponding MetaData 'xml' file (with extension .xmlc) for every created DOM class. For example: file 'Welcome.html' will, as result of XMLC compilation (with Deferred Parsing option on), caused generation of 'WelcomeHTML.java' (DOM class) and 'WelcomeHTML.xmlc' (MetaData) files.

- Unfortunately, deployment of 'war' archive, built in this manner, still won't work in servlet container. The 'war' file, created by 'build.xml' files, which are originally created by 'appwizard', are located in:

```
<app_root>/presentation/bin/presentation/webapps/<app_name>.war
```

The 'war' file, in its inner organisation, originally contains Enhydra application placed in jar file as:

```
<WEB-INF>/lib/<app_name>.jar
```

The Enhydra application, which will use the Deferred Parsing, must have all its classes placed out of 'jar' file in a regular directory structure, under the 'classes' directory. Also, that directory must contain, beside each generated DOM class, its corresponding MetaData 'xml' file and its origin HTML file. For example: beside 'WelcomeHTML.class' file, in its package, the 'WelcomeHTML.xmlc' and 'Welcome.html' files must be placed. So, the 'war' file instead lib directory must have all application classes (and MetaData and HTML files) placed in the structure:

```
<WEB-INF>/classes/<...app_structure...>
```

To fit the above mentioned requirements, the changing of 'war' file can be done manually, or by changing build.xml files originally created by 'appwizard', so that the invoking of 'enhydra-ant' command will generate the well formed 'war' file.

Note that 'war' file is suitable for autoload process in servlet container. Also, the application can be deployed without wrapping in archive, but in appropriate web application structure.

- The last step will be deployment and start of Enhydra application in servlet container. The Deferred Parsing can be tested by changing and saving any 'HTML' file from application deploy classes directory. The changes can be noticed by visiting changed page, without stopping the application.

Auto Reloading

The process of obtaining Enhydra application to run with Auto Reloading has much similarity with the previously described Deferred Parsing. The following preparation actions must be done:

- XMLCFactory must be used to instantiate pages in your Java code, i.e.

```
 WelcomeHTML welcome =
  (WelcomeHTML)comms.xmlcFactory.create(WelcomeHTML.class);
```

  The creation of Java objects with DOM representation of HTML pages in Enhydra presentation objects, in the case of the Deferred Parsing and Automatic Reloading must be done only by create method of the xmlcFactory class (like in the example above). The creation of instances with the Java key word 'new' must not be performed. This way of instances creation (with 'new') can be used in applications in which Deferred Parsing and Automatic Reloading are not used.

- Appropriate Auto Reload parameter must be set in application configuration file (*<app_name>.conf* or *web.xml* file depending on which way of application parameters reading is chosen). If *<app_name>.conf* file is used, the parameter should be set as below:

```
Server.AutoReload
  = true
```

  while the same parameter in case of *web.xml* should be defined as 'env-entry' tag, shown below:

```
<env-entry>
  <env-entry-name>Server/AutoReload</env-entry-name>
  <env-entry-value>true</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
```

  Note that the configuration file that will be used for reading application parameters is defined in corresponding application web.xml file (parameters 'ConfFile' and 'ConfFileClass').

  In the case when an application project has been created by 'appwizard', the two application configuration template files should be edited, in order to define Auto Reload via <app_name>.conf file:

```
<app_root>/presentation/resource/appwizard/conf/<app_name>.conf.in
  <app_root>/application/resource/appwizard/conf/<app_name>.conf.in
```

  Also, the two web.xml files should be edited in order to set Auto Reload parameter, via application web.xml file:

```
<app_root>/presentation/resource/appwizard/WEB-INF/web.xml
  <app_root>/application/resource/appwizard/WEB-INF/web.xml
```

- Before any XMLC parsing, in each subdirectory which contains resources (directories with HTML files), file options.xmlc must be edited with following parameter:

```
-generate
```

```
  both
```

This parameter suggests, to later invoked XMLC parser, that resources (HTML files) should be parsed into DOM objects for Auto Reloading usage.

- After all preparation acts mentioned in the previous tips were done, the building of application project prepared for Auto Reloading can be done.

  In the case when an application project has been made by 'appwizard', the invoking of command on the command line:

  ```
  enhydra-ant
  ```

  will start ant based build process, which has as one of its tasks XMLC parsing of the resources. The results of XMLC parsing are located in directory:

  ```
  <app_root>/presentation/src-generated
  ```

  and they consist of two Java classes (interface and its implementation) which represent DOM for each HTML resource file. For example: file 'Welcome.html' will, as result of XMLC compilation (with Auto Reloading option on), caused generation of 'WelcomeHTML.java' (interface) and 'WelcomeHTMLImpl.java' (implementation of interface) files.

- Unfortunately, deployment of 'war' archive, build in this manner, still won't work in servlet container. The 'war' file, created by 'build.xml' files, which are originally created by 'appwizard', are located in:

  ```
  <app_root>/presentation/bin/presentation/webapps/<app_name>.war
  ```

  The 'war' file, in its inner organisation, originally contains Enhydra application placed in jar file as:

  ```
  <WEB-INF>/lib/<app_name>.jar
  ```

  The Enhydra application, which will use the Automatic Reloading, must has all its classes placed out of 'jar' file in a regular directory structure, under the 'classes' directory. Also, that directory must contain both generated DOM classes (interface and its implementation). For example: beside 'WelcomeHTML.class' file, in its package, the 'WelcomeHTMLImpl.class' file must be placed. So, the 'war' file instead lib directory with jar file must have all application classes placed in structure:

  ```
  <WEB-INF>/classes/<...app_structure...>
  ```

  To fit the above mentioned requirements, the changing of 'war' file can be done manually, or by changing build.xml files originally created by 'appwizard', so that the invoking of 'enhydra-ant' command will generate the well formed 'war' file.

  Note that 'war' file is suitable for autoload process in servlet container. Also, the application can be deployed without wrapping in archive, but in appropriate web application structure.

- The last step will be deployment and start of Enhydra application in servlet container. The Automatic Reload can be tested by changing and saving any 'HTML' file or class in application project. The project must be rebuilt, and after that the changed classes should be copied to application deploy classes directory. The changes can be noticed by visiting changed page, without stopping the application.

Deferred Parsing and Auto Reload Summary

There are three facts you need to understand in order to make Deferred Parsing or Auto Reload work

- The application can not be deployed in a jar file. Enhydra applications are typically deployed as single Java Archive Files with all of the necessary classes. However, updating a jar file at runtime with new classes does not make sense, so the application's classes must be in a regular directory structure. In the case of Deferred Parsing, the HTML files and generated MetaData files must be in the same directories as the corresponding compiled Java classes.
- Pages must be instantiated with the xmlcFactory. In general, XMLC page objects can be instantiated with either 'new' or the xmlcFactory as shown in the following code:

  ```
  WelcomeHTML welcome = new WelcomeHTML();
  ```

  or

  ```
  WelcomeHTML welcome =
   (WelcomeHTML)comms.xmlcFactory.create(WelcomeHTML.class);
  ```

  However, with Deferred Parsing (also with Auto Reload), you must instantiate a page with the create method of xmlcFactory. The factory does the work of checking the timestamps on the files and dynamically recompiling an HTML page if it has been changed. It's a better programming practice always to use xmlcFactory to instantiate pages. By default, it simply does a 'new' on the page, and the use of it gives you the option of turning on Deferred Parsing or other factory features at a later date.
- HTML file can be updated at runtime via Deferred Parsing, while via Auto Reloading updated classes can be loaded in runtime of application.

  Consider the following :

  - The source HTML file is compiled . . .
  - into a Java class file that is . . .
  - loaded by the server at runtime.

To replace the Java class representing a page at runtime, you can replace either the original HTML file (in Deferred Parsing) or the class file compiled from the page (in Auto Reloading). For example, a running application has loaded 'WelcomeHTMLImpl.class'. To change the look of the page it generates, you could either replace the file 'Welcome.html' (in Deferred Parsing) or recompile 'Welcome.html' yourself and replace the files 'WelcomeHTML.class' and 'WelcomeHTMLImpl.class' (in Auto Reloading).

The options you specify in options.xmlc and Enhydra configuration file (<*app_name*>*.conf* or *web.xml* file) reflect these two possibilities. To only reload classes, invoke XMLC with the '-generate both' option. This generates both an interface and an implementation class. To dynamically load HTML, use the XMLC option '-for-deferred-parsing'. This option tells XMLC to generate an implementation class, and *.xmlc file for each HTML page. The *.xmlc file includes the options used to compile the page so the same options are called when it is dynamically recompiled. The *.xmlc file contains options written in the MetaData xml language.

There are also options set in Enhydra application configuration file (<*app_name*>*.conf* or *web.xml* file) for turning on Auto Reloading and Deferred Parsing at runtime. Specify:

```
Server.AutoReload =
  true
```

to enable class reloading and

```
Server.XMLC.DeferredParsing = true
```

to enable deferred parsing.

Note that the Auto Reloading and Deferred Parsing capabilities come at the cost of checking the timestamps on either the class file or the class and HTML files each time the page is instantiated at runtime.

Memory Persistence

Making an application to work with Memory Persistence option turned on is much easier than it was with Deferred Parsing or Auto Reloading:

- Appropriate Memory Persistent parameter must be set in application configuration file (*<app_name>.conf* or *web.xml* file depending on which way of application parameters reading is chosen). If *<app_name>.conf* is used, the parameter should be set as below:

  ```
  SessionManager.MemoryPersistence
    = true
  ```

  while the same parameter in case of*web.xml* file should be defined as 'env-entry' tag, shown below:

  ```
  <env-entry>
    <env-entry-name>SessionManager/MemoryPersistence</env-entry-name>
    <env-entry-value>true</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
  ```

  Note that the configuration file that will be used for reading application parameters is defined in corresponding application web.xml file (parameters 'ConfFile' and 'ConfFileClass').

  In the case when an application project has been made by 'appwizard', the two application configuration template files should be edited, in order to define Memory Persistence via <app_name>.conf file:

  ```
  <app_root>/presentation/resource/appwizard/conf/<app_name>.conf.in
    <app_root>/application/resource/appwizard/conf/<app_name>.conf.in
  ```

  Also, the two web.xml files should be edited in order to set Memory Persistence parameter, via application web.xml file:

  ```
  <app_root>/presentation/resource/appwizard/WEB-INF/web.xml
    <app_root>/application/resource/appwizard/WEB-INF/web.xml
  ```

- After all preparation acts mentioned in the previous tips were done, the building of application project prepared for Memory Persistence can be done.

  In the case when an application project has been made by 'appwizard', the invoking of command on the command line:

  ```
  enhydra-ant
  ```

will start ant based build process.

- The 'war' file, created by 'build.xml' files originally created by 'appwizard', are located in:

```
<app_root>/presentation/bin/webapps/autoload/<app_name>.war
```

  The created 'war' can be used as is, without any changing.

- The last step will be deployment and start of Enhydra application in servlet container. The aim of Memory Persistence is to keep session data stored in memory while application is stopped and restarted again. To test this possibility, appropriate application must be created, which should save some data in session object, and than reuse that data again. That means that application, originally created by 'appwizard', should be slightly changed so that the effect of turning on Memory Persistence option will become visible.