

ImageCaptchaFilter User Manual

Version : 1.0
Date : 25/01/2004
Author : sbrunot@octo.com

Content :

What is ImageCaptchaFilter ?	3
How to add captchas to an existing web app ?.....	4
Modify your entry form so that it displays the captcha.....	6
Add ImageCaptchaFilter to the web app and parameterize it	10
How to manage ImageCaptchaFilter at runtime ?.....	15

Figures :

Figure 1 : original J2EE petstore's "Add new user" form	4
Figure 2 : J2EE petstore's "Add new user" form with a captcha.....	5
Figure 3 : declaration and parameterization of ImageCaptchaFilter in the petstore's web.xml	13

Tables :

Table 1 : ImageCaptchaFilter's parameters reference table	12
Table 2 : ImageCaptchaFilterMBean attributes	17

What is ImageCaptchaFilter ?

ImageCaptchaFilter is a J2EE Filter provided by the jcaptcha-j2ee library (<http://jcaptcha.sourceforge.net>). It is designed to help you add visual captchas (see <http://www.captcha.net> if you don't know yet what a captcha is) to the entry forms of your existing MVC web applications.

Current features of ImageCaptchaFilter are:

- ✍ Generation and rendering of Captcha images (as JPEG) at runtime for inclusion in an existing entry form. ImageCaptchaFilter use a ImageCaptchaEngine (see jcaptcha-core documentation at <http://jcaptcha.sourceforge.net>) to generate captchas : a simple one, com.octo.captcha.image.gimpy.SimpleGimpyEngine, which displays a random string made up with characters A,B,C,D and E, is provided with jcaptcha-j2ee but you can author your own or use one of those provided with jcaptcha-samples ;
- ✍ Verification of the HTTP client entry in response to the challenge displayed as an image in the entry form:
 - ✍ Redirection of the HTTP client to an error page if the captcha challenge is not passed ;
 - ✍ Transparent follow up of the request to the web application, without any information concerning the captcha, if the challenge is successfully passed ;
- ✍ Many forms can be protected by captchas in the same web application, each one having its own error redirection page ;
- ✍ Monitoring via JMX :
 - ✍ Number of captcha generated since the filter is up;
 - ✍ Number of captcha which challenge has been successfully passed;
 - ✍ Number of captcha the client failed to pass the challenge;
 - ✍ Performance tuning;
- ✍ Administration via JMX :
 - ✍ Hot modification of the ImageCaptchaEngine implementation used by the filter;
 - ✍ Performance tuning.
- ✍ Security audit: generation of WARN messages providing client IP address when the ImageCaptchaFilter receives incorrect requests.

How to add captchas to an existing web app ?

This section of the manual is a tutorial on adding captchas to an existing webapp's form using the ImageCaptchaFilter component. The Sun J2EE petstore application version 1.3.2 is used here as a sample. It can be downloaded at <http://java.sun.com/developer/releases/petstore/>. For more information about the Sun J2EE petstore application version 1.3.2, have a look at <http://java.sun.com/blueprints/code/jps132/docs/index.html>.

In this tutorial, we'll add a captcha to the petstore form provided to declare a new user to the system (let's name it the "Add new user" form). This is a typical usage of captchas: preventing robots from massively declaring fake users to the system (see <http://login.yahoo.com/> and follow the "Sign up now" link to see a real world example of such usage). Figure 1 shows how the "Add new user form" looks like in the original J2EE petstore:

A screenshot of the "Sign In" form from the original J2EE petstore. On the left, there is a "Pets" sidebar with links for Birds, Cats, Dogs, Fish, and Reptiles. The main form is titled "Sign In" and asks "Are you a returning customer?". It is split into two columns. The left column is for "Yes." and contains fields for "User Name:" (with "j2ee" entered) and "Password:" (with four dots), a "Sign In" button, and a "Remember My User Name" checkbox. The right column is for "No. I would like to sign up for an account." and contains fields for "User Name:", "Password:", and "Password (Repeat):", along with a "Create New Account" button.

Figure 1 : original J2EE petstore's "Add new user" form

Assuming you've deployed the original petstore on a local j2ee server which listen for incoming HTTP requests on port 8000, the original "Add new user" form can be reached at http://localhost:8000/petstore/signon_welcome.screen.

What we are going to do in this tutorial is to add a gimpy captcha before the "Create New Account" button : an image will be displaying a random string, which the user will be requested to spell in a dedicated text field. Figure 2 shows how the form will look with our captcha :

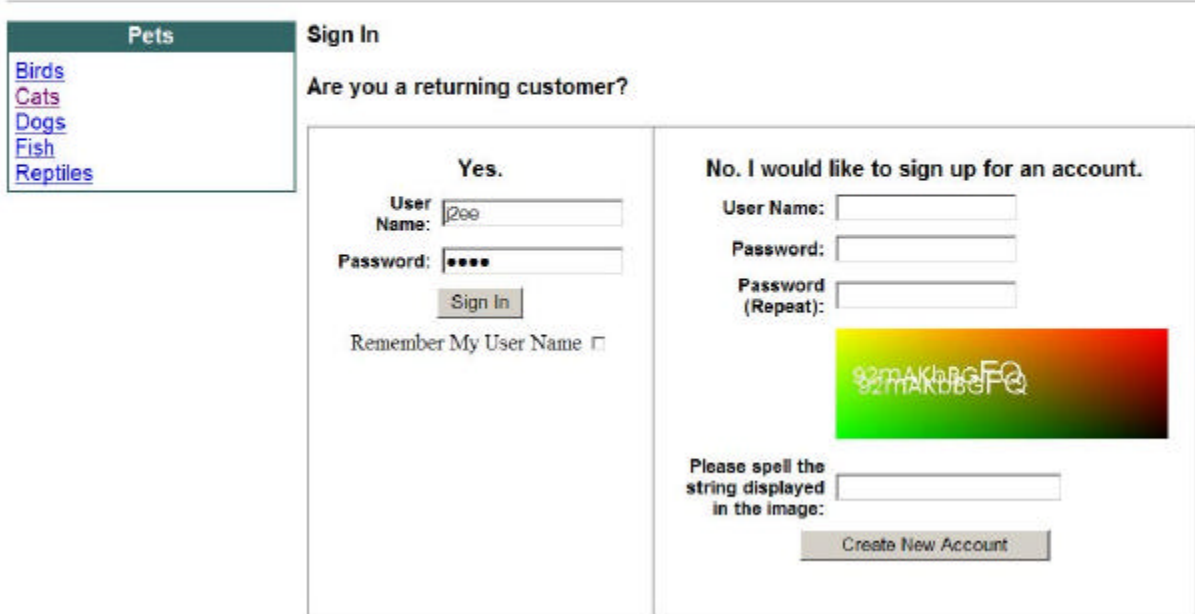


Figure 2 : J2EE petstore's "Add new user" form with a captcha

With a captcha in it, the user manual for the "Add new user" form is:

- ✍ Type user information (name and password) in the dedicated fields, as it had to be done in the original form ;
- ✍ Type the string displayed by the captcha image in the text field labelled "Please spell the string displayed in the image" (the string displayed in Figure 2 being "92mAKbBGfQ") ;
- ✍ Press the "Create New Account" button to ask for the new account creation, as it had to be done in the original form.

When the modified petstore receives such a "create new account" request, we want it to behave like this:

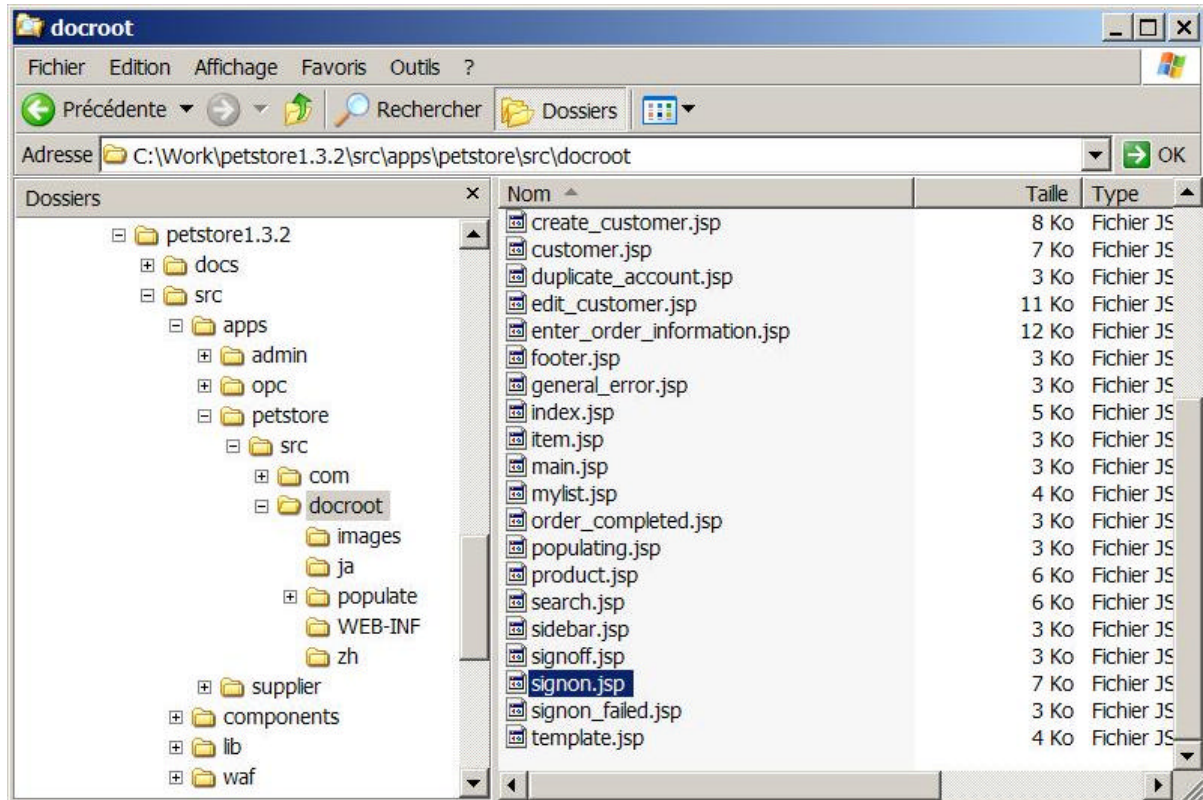
- ✍ If the string "entered by the user" is the one displayed by the captcha image, process exactly as it was done in the original petstore application (we don't even have to know what this processing was) ;
- ✍ If not, re display the entry form.

To achieve those new requirements with ImageCaptchaFilter, we just have to:

- ✍ Modify the entry form so that it displays the captcha, which implies authoring some jsp code ;
- ✍ Add the ImageCaptchaFilter to the J2EE petstore web app and parameterize it, which implies adding some extra libraries in the petstore WEB-INF\lib subdirectory and extra deployment descriptors in its web.xml.

Modify your entry form so that it displays the captcha

The jsp code we have to modify is in the *signon.jsp* source file, which is stored in the *src/apps/petstore/src/docroot* subdirectory of the J2EE petstore sources directory:



WARNING : because of the petstore design, we also have to modify the *signon.jsp* files which are stored in the *src/apps/petstore/src/docroot/ja/* and *src/apps/petstore/src/docroot/zh/* subdirectories to provide captcha for the Japanese and Chinese users. If we don't, the new petstore won't be protected from robots massive fake users creation.

Here is the jsp code that produce the “Add new user” form (lines 131 to 183 of *signon.jsp*) :

```
<waf:form name="newcustomer" action="createuser.do" method="POST">
<table cellpadding="5" cellspacing="0" border="0">
<tr>
<td class="petstore" align="center" colspan="2">
<b>No. I would like to sign up for an account.</b>
</td>
</tr>
<tr>
<td class="petstore_form" align="right">
<b>User Name:</b>
</td>
<td class="petstore_form">
<waf:input cssClass="petstore_form"
type="text"
size="15"
validation="validation"
name="j_username"/>
</td>
</tr>
```

```

</tr>
<tr>
<td class="petstore_form" align="right">
<b>Password:</b>
</td>
<td class="petstore_form">
<waf:input cssClass="petstore_form"
type="password"
size="15"
validation="validation"
name="j_password"/>
</td>
</tr>
<tr>
<td class="petstore_form" align="right">
<b>Password (Repeat):</b>
</td>
<td class="petstore_form">
<waf:input cssClass="petstore_form"
type="password"
size="15"
validation="validation"
name="j_password_2"/>
</td>
</tr>
<tr>
<td align="center" colspan="2">
<input class="petstore_form"
name="submit"
type="submit"
value="Create New Account"/>
</td>
</tr>
</table>
</waf:form>

```

Here is the modified jsp after adding a captcha (the extra code is displayed in bold) :

```

<waf:form name="newcustomer" action="createuser.do" method="POST">
<table cellpadding="5" cellspacing="0" border="0">
<tr>
<td class="petstore" align="center" colspan="2">
<b>No. I would like to sign up for an account.</b>
</td>
</tr>
<tr>
<td class="petstore_form" align="right">
<b>User Name:</b>
</td>
<td class="petstore_form">
<waf:input cssClass="petstore_form"
type="text"
size="15"
validation="validation"
name="j_username"/>
</td>
</tr>
<tr>
<td class="petstore_form" align="right">
<b>Password:</b>
</td>
<td class="petstore_form">
<waf:input cssClass="petstore_form"

```

```

                type="password"
                size="15"
                validation="validation"
                name="j_password"/>
            </td>
        </tr>
        <tr>
            <td class="petstore_form" align="right">
                <b>Password (Repeat):</b>
            </td>
            <td class="petstore_form">
                <waf:input cssClass="petstore_form"
                    type="password"
                    size="15"
                    validation="validation"
                    name="j_password_2"/>
            </td>
        </tr>
        <!-- EXTRA CODE TO ADD CAPTCHA BEGINS HERE -->
        <tr>
            <%
                int captchaID = (int) (Math.random() * Integer.MAX_VALUE);
                String captchaRenderingURL = "/captcha/render.do";
                String captchaIDParameterName = "captcha_id";
                String renderURL = "/petstore" + captchaRenderingURL + "?" + captchaIDParameterName + "="
                    + captchaID;
            <%>
            <td>
            </td>
            <td>
                <img src=<%=renderURL%>>
                <input type="hidden" name="<%=captchaIDParameterName%>" value="<%=captchaID%>">
            </td>
        </tr>
        <tr>
            <td class="petstore_form" align="right">
                <b>Please spell the string displayed in the image:</b>
            </td>
            <td class="petstore_form">
                <%
                    String captchaChallengeResponseParameterName = "challenge_response";
                <%>
                <waf:input cssClass="petstore_form"
                    type="text"
                    name="<%=captchaChallengeResponseParameterName%>" />
            </td>
        </tr>
        <!-- EXTRA CODE TO ADD CAPTCHA ENDS HERE -->
        <tr>
            <td align="center" colspan="2">
                <input class="petstore_form"
                    name="submit"
                    type="submit"
                    value="Create New Account"/>
            </td>
        </tr>
    </table>
</waf:form>

```

The first part of the extra code addresses the generation of a captcha by the ImageCaptchaFilter and the rendering of his challenge as a jpeg image in the “Create new user” form:

```
<%
```



```

int captchaID = (int) (Math.random() * Integer.MAX_VALUE);
String captchaRenderingURL = "/captcha/render.do";
String captchaIDParameterName = "captcha_id";
String renderURL = "/petstore" + captchaRenderingURL + "?" + captchaIDParameterName + "=" + captchaID;
%>
[.../
<img src=<%=renderURL%>>

```

They are more elegant ways to code this (using a taglib instead of “ugly” jsp code, etc...), but the principle here is to keep it simple so that the mechanism remains easily understandable:

- ✍ First of all, we generate an id for the captcha we want the ImageCaptchaFilter to generate. The “generate and render as jpeg” function of CaptchaFiler associates an id with each captcha instance it creates and this id must be provided by the filter client. There are three simple rules to follow for generating this id :
 - ✍ It is a string, which is passed to the ImageCaptchaFilter as an HTTP request parameter ;
 - ✍ It has a maximum length. As this maximum length is a parameter of the ImageCaptchaFilter (“CaptchaIDMaxLength” : see Table 1 for more details about this), you can choose whichever maximum length you want but you must ensure that the length of the id that is created in the jsp is less than it. If not, any request done to the ImageCaptchaFilter providing this id will return a 404 and, as a result, no image will be displayed by the form ;
 - ✍ It must be a different one for each generated HTML page. ImageCaptchaFilter associates a unique captcha for a given id: if you ask for the generation of a captcha providing id id_1 , and then ask for another captcha with the same id id_1 while the first one hasn’t been challenged yet, the second generated captcha will replace the first one (so your first user, in order to prove he is a human being, will have to pass the second captcha challenge while being submitted the first one by the form... hope this is clear enough).

In our example here, we generated a random id that is an integer (as a string) between 0 and $2^{31} - 1$, so the maximum length of this id is 10.

- ✍ Then we add an tag which HREF URL means to the ImageCaptchaFilter “generate a captcha and render it as jpeg in the response”. This URL base is a parameter of the ImageCaptchaFilter (“RenderURL” : see Table 1 for more details about this). The name of the HTTP request parameter we use to transmit the captcha id is also a parameter of the ImageCaptchaFilter (“CaptchaIDParameterName” : see Table 1 for more details about this) ;

The second part of the extra code addresses the verification by the ImageCaptchaFilter of the user’s response to the captcha challenge:

```

<input type="hidden" name="<%=captchaIDParameterName%>" value="<%=captchaID%>">
[.../
<td class="petstore_form" align="right">
  <b>Please spell the string displayed in the image:</b>
</td>
<td class="petstore_form">
  <%
    String captchaChallengeResponseParameterName = "challenge_response";
    %>
  <waf:input cssClass="petstore_form"
    type="text"
    name="<%=captchaChallengeResponseParameterName%>" />
  </td>

```

- ✍ First, we add a hidden input that stores the captcha id so that it is transmitted to the ImageCaptchaFilter on form validation. The name of this hidden input is the name of the HTTP request parameter also used for the generation and rendering of the captcha (see above);
- ✍ Then we add the text field entry in which the user must spell the string displayed by the captcha image. The name of the text field (which is the name of the HTTP request parameter used to transmit the user's response to the ImageCaptchaFilter) is a parameter of the ImageCaptchaFilter ("CaptchaChallengeResponseParameterName" : see Table 1 for more details about this).

Add ImageCaptchaFilter to the web app and parameterize it

To add the captcha filter to the web app (petstore in our example) and parameterize it, we just have to:

- ✍ Include in the WEB-INF\lib subdirectory of the web application the jcaptcha-core-version.jar and jcaptcha-j2ee-version.jar libraries provided with the JCaptcha distribution. As the filter provides JMX management capabilities and rely on Jakarta commons-logging (<http://jakarta.apache.org/commons-logging>) for its logs, you must ensure that the corresponding libraries are present in the classpath of your application server. If not, and it is the case with the j2ee 1.3.1 RI, bundle them in your web app copying the corresponding jars in the WEB-INF\lib (for JMX, you can download the reference implementation at <http://java.sun.com/products/JavaManagement>) ;
- ✍ Modify the web.xml to declare the filter. Keep in mind that the ImageCaptchaFilter must be the first filter declared in web.xml.

Please consult the documentation provided with J2EE petstore 1.3.1 for details about how to modify its build script and rebuild it.

Here is the reference for all the parameters that should be set to configure ImageCaptchaFilter:

Parameter	Description
ImageCaptchaEngineClass	<p>The ImageCaptchaEngine concrete class used by ImageCaptchaFilter to create captchas (see jcaptcha-core documentation to get more information about how to provide your own concrete class : http://jcaptcha.sourceforge.net).</p> <p>Example : to use the simple gimpy engine provided with jcaptcha-j2ee, declare this simple engine class :</p> <pre><init-param> <param-name>ImageCaptchaEngineClass</param-name> <param-value> com.octo.captcha.image.gimpy.SimpleGimpyEngine </param-value> </init-param></pre>
CaptchaRenderingURL	<p>This is the URL used to request generation and rendering as jpeg of a new captcha. It is relative to the web application root and must begin with a "/". A Call to this URL providing under parameter <i>CaptchaIDParameterName</i> (see below) an id which length is between 1 and <i>CaptchaIDMaxLength</i> (see below) returns a captcha challenge as a jpeg image.</p>

	<p>Example : if you want that a call to http://server:port/WebApp/captcha/render.do?CaptchaIDParameterName=ID creates a new captcha and returns its challenge as a jpeg image, use :</p> <pre> <init-param> <param-name>CaptchaRenderingURL</param-name> <param-value>/captcha/render.do</param-value> </init-param> </pre>
CaptchaVerificationURLs	<p>This is the “;” separated values list of the captcha-ized form’s action URL. Each one is relative to the web application root and must begin with a “/”. The ImageCaptchaFilter intercepts all requests to those URLs and search for a captcha id and a challenge response in it (respectively the <i>CaptchaIDParameterName</i> and the <i>CaptchaChallengeResponseParameterName</i> request parameters). If those parameters are not included in the request, or if a captcha id is present but doesn’t match any captcha previously generated and not yet checked, the filter sends back an HTTP 404 error and a security event is logged. If the parameters are present, the filter verifies the response to the captcha challenge. If the response is correct, the request is cleaned of the captcha specific parameters and then forwarded to the component that processes it in the original web application (the original web application being the web application without ImageCaptchaFilter). If the response is not correct, the client is redirected to an URL : <i>CaptchaErrorURLs</i> is the “;” separated values list of URL the client is redirected to (the first is used with the first verification URL, the second with the second one, etc...).</p> <p>Example : if a captcha verification must be done for any request to http://server:port/WebApp/usecaseone/step2.do and http://server:port/WebApp/usecasetwo/step2.do, and that the “user” must be redirected to http://server:port/WebApp/error1.jsp if he fails the captcha challenge for the first URL, and http://server:port/WebApp/error2.jsp if he fails the captcha challenge for the second URL :</p> <pre> <init-param> <param-name>CaptchaVerificationURLs</param-name> <param-value> /usecaseone/step2.do;/usecasetwo/step2.do </param-value> </init-param> <init-param> <param-name>CaptchaErrorURLs</param-name> <param-value>error1.jsp;error2.jsp</param-value> </init-param> </pre>
CaptchaErrorURLs	See <i>CaptchaVerificationURLs</i> above
CaptchaIDParameterName	<p>This is the name of the HTTP request parameter used to provide a captcha id for captcha generation and verification (see <i>CaptchaRenderingURL</i> and <i>CaptchaVerificationURLs</i> above).</p> <p>Example : HTTP request parameter captcha_id contains the captcha id :</p> <pre> <init-param> <param-name>CaptchaIDParameterName</param-name> <param-value>captcha_id</param-value> </init-param> </pre>
CaptchaIDMaxLength	<p>This is the maximum length of a captcha id. If the filter intercepts a generation or verification request which provides an id which length majors this value, it will return an HTTP 404 Error and will generate a security log providing the caller’s IP address.</p> <p>Example : Captcha id must be 10 character longs max :</p> <pre> <init-param> </pre>

	<pre> <param-name>CaptchaIDMaxLength</param-name> <param-value>10</param-value> </init-param> </pre>
CaptchaChallengeResponseParameterName	<p>This is the name of the HTTP request parameter used to provide a user's response to a captcha challenge for verification (see <i>CaptchaVerificationURLs</i> above).</p> <p>Exemple : HTTP request parameter challenge_response contains the user's challenge response :</p> <pre> <init-param> <param-name> CaptchaChallengeResponseParameterName </param-name> <param-value>challenge_response</param-value> </init-param> </pre>
CaptchaInternalStoreSize	<p>When the filter generates a captcha, it stores it in an internal store. When it receives a verification request for a captcha, it delete it from the internal store. When the internal store is full, the filter try to remove the captchas which time to live (<i>CaptchaTimeToLive</i>, the time to live in milliseconds) is over. If it can't remove any captcha, it didn't generate a new one and the user must reload the form to try to get a captcha. CaptchaInternalStoreSize is the size of the internal store.</p> <p>Example : 1000 captchas maximum in the internal store with a time to live of 2 minutes (120000 milliseconds) :</p> <pre> <init-param> <param-name>CaptchaInternalStoreSize</param-name> <param-value>1000</param-value> </init-param> <init-param> <param-name>CaptchaTimeToLive</param-name> <param-value>120000</param-value> </init-param> </pre>
CaptchaTimeToLive	See <i>CaptchaInternalStoreSize</i> above
RegisterToMBeanServer	<p>This is a boolean value : if "true", the CaptchaFilter will try to register itself to a JMX server during initialization (use "true" with JBoss, for example). If "false", it will not.</p> <p>Example : try to register with a JMX server :</p> <pre> <init-param> <param-name>RegisterToMBeanServer</param-name> <param-value>true</param-value> </init-param> </pre>

Table 1 : ImageCaptchaFilter's parameters reference table

Here is the modified petstore web.xml after declaring and parameterizing ImageCaptchaFilter (the extra deployment descriptors are displayed in **bold**):

```

[ ... ]

<web-app>
  <display-name>PetStoreWAR</display-name>
  <description>Web Tier DD for the PetStore application</description>
  <!-- Captcha Filter Declaration Start -->

```

```

<filter>
  <filter-name>ImageCaptchaFilter</filter-name>
  <filter-class>com.octo.captcha.j2ee.servlet.ImageCaptchaFilter</filter-class>
  <init-param>
    <!-- The image engine used to generate captchas -->
    <param-name>ImageCaptchaEngineClass</param-name>
    <param-value>com.octo.captcha.image.gimpy.SimpleGimpyEngine</param-value>
  </init-param>
  <init-param>
    <!-- the URL to call for captcha generation and JPEG rendering -->
    <param-name>CaptchaRenderingURL</param-name>
    <param-value>/captcha/render.do</param-value>
  </init-param>
  <init-param>
    <!--The URL where "Add new user" form data are posted to -->
    <param-name>CaptchaVerificationURLs</param-name>
    <param-value>/createuser.do</param-value>
  </init-param>
  <init-param>
    <!--The URL of the "Add new user" form (redirect to this form if the
      captcha challenge is not resolved) -->
    <param-name>CaptchaErrorURLs</param-name>
    <param-value>/signon_welcome.screen</param-value>
  </init-param>
  <init-param>
    <param-name>CaptchaIDParameterName</param-name>
    <param-value>captcha_id</param-value>
  </init-param>
  <init-param>
    <param-name>CaptchaIDMaxLength</param-name>
    <param-value>10</param-value>
  </init-param>
  <init-param>
    <param-name>CaptchaChallengeResponseParameterName</param-name>
    <param-value>challenge_response</param-value>
  </init-param>
  <init-param>
    <param-name>CaptchaInternalStoreSize</param-name>
    <param-value>100</param-value>
  </init-param>
  <init-param>
    <param-name>CaptchaTimeToLive</param-name>
    <param-value>120000</param-value>
  </init-param>
  <init-param>
    <param-name>RegisterToMBeanServer</param-name>
    <param-value>false</param-value>
  </init-param>
</filter>
<!-- Captcha Filter Declaration End -->
<!-- Encoding Filter Declaration Start -->

[...]

<!-- Signon Filter Declaration End -->
<!-- Captcha Filter Mapping Start -->
<filter-mapping>
  <filter-name>ImageCaptchaFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- Captcha Filter Mapping End -->
<!-- ComponentManager Listener -->

[...]

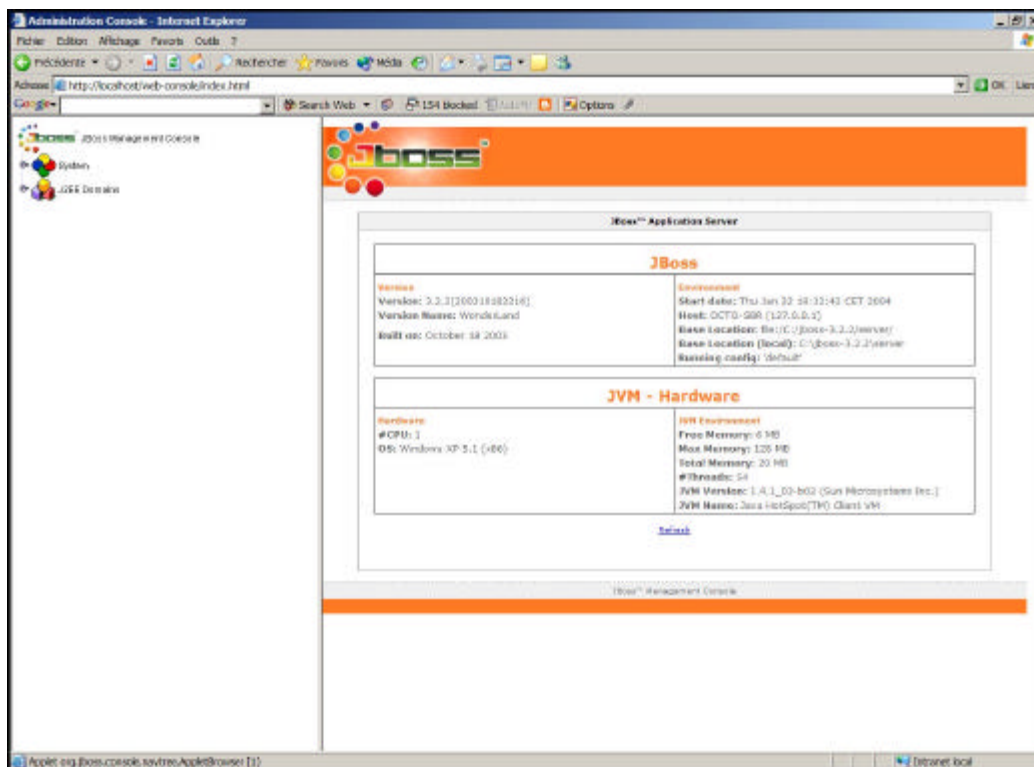
```

Figure 3 : declaration and parameterization of ImageCaptchaFilter in the petstore's web.xml

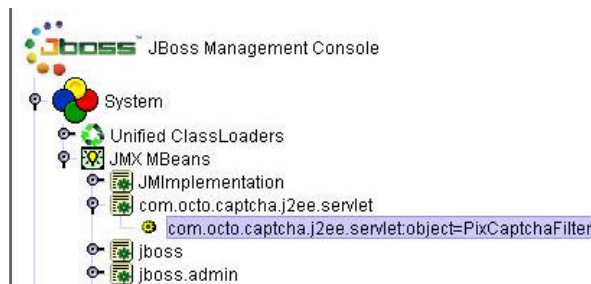
How to manage ImageCaptchaFilter at runtime ?

To manage ImageCaptchaFilter at runtime, you must deploy the web application that uses it in a JMX enabled J2EE Server (JBoss 3.2.x, for example). The ImageCaptchaFilter MBean is named “com.octo.captcha.j2ee.servlet:object=ImageCaptchaFilter”. Here is an example of how to manage it using the JBoss web-console:

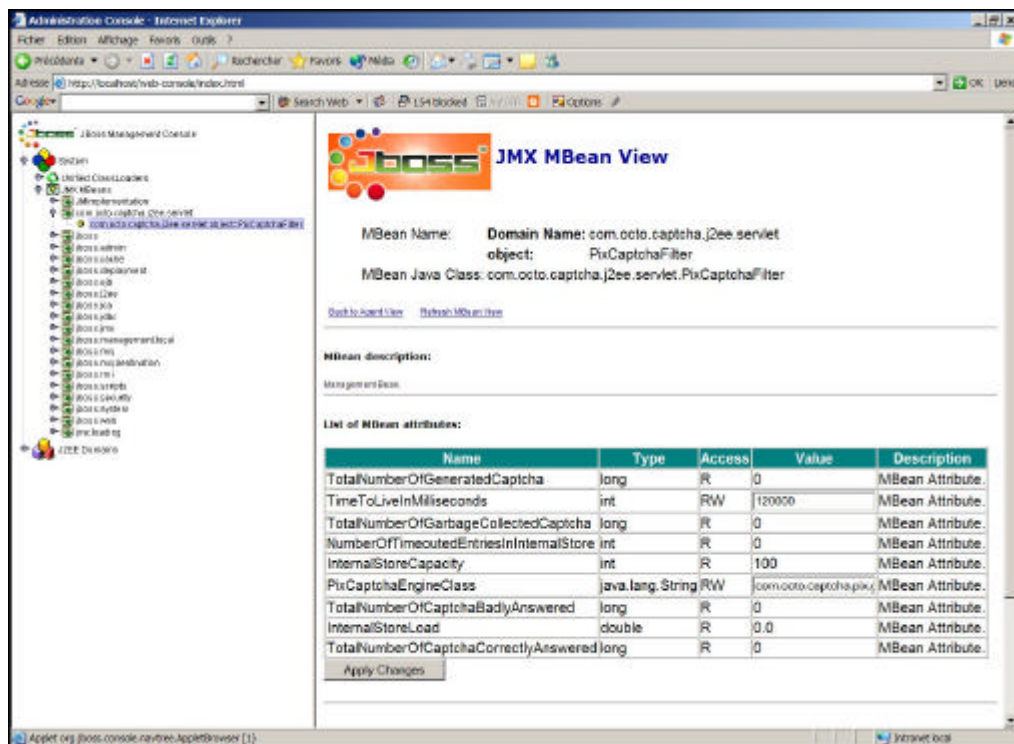
- 1) Connect to the JBoss web-console (which URL is usually <http://server:port/web-console>) :



- 2) Click the “System” node in the left panel and search for the com.octo.captcha.j2ee.servlet:object=ImageCaptchaFilter MBean :



- 3) Click on the MBean name to display the JMX Mbean view in the right panel :



- 4) The attributes for this MBean are :

Name	Type	Access	Description
ImageCaptchaEngineClass	String	RW	The fully qualified class name of the concrete ImageCaptchaEngine used by the filter
InternalStoreCapacity	int	R	The capacity of the internal store
TimeToLiveInMilliseconds	int	RW	The time to live for entries in the internal store
InternalStoreLoad	double	R	The load of the internal store (number of current entries / capacity of the store)
NumberOfTimeoutedEntriesInInternalStore	int	R	The number of timeouted entries in the internal store.
TotalNumberOfGeneratedCaptcha	long	R	The number of captcha generated since the Filter is up. WARNING : this value won't be significant if the total number is > Long.MAX_VALUE
TotalNumberOfCaptchaCorrectlyAnswered	long	R	The number of captcha for wich the challenge was correctly answered since the Filter is up. WARNING : this value won't be significant if the total number is > Long.MAX_VALUE

TotalNumberOfCaptchaBadlyAnswered	long	R	The number of captcha for wich the challenge was badly answered since the Filter is up. WARNING : this value won't be significant if the total number is > Long.MAX_VALUE
TotalNumberOfGarbageCollectedCaptcha	long	R	The number of captcha garbage collected since the Filter is up. WARNING : this value won't be significant if the total number is > Long.MAX_VALUE

Table 2 : ImageCaptchaFilterMBean attributes

- 5) The MBean also got one method, void garbageCollectInternalStore(), that removes all timeouted captchas from the internal store :

List of MBean operations:

void garbageCollectInternalStore()

MBean Operation.

Invoke