RichFaces Performance and Best Practices

- Re-Rendering, Regions and Partial Updates
- Using Tools
- Queues
- Reducing Response Size

RichFaces Performance is Complex Issue

- There are many factors that must be considered when addressing RichFaces performance, including:
    - Browser JavaScript execution speed
    - Browser DOM update speed
    - Browser rendering speed
    - Network bandwidth
    - Server utilization and load

General RichFaces Good Practices

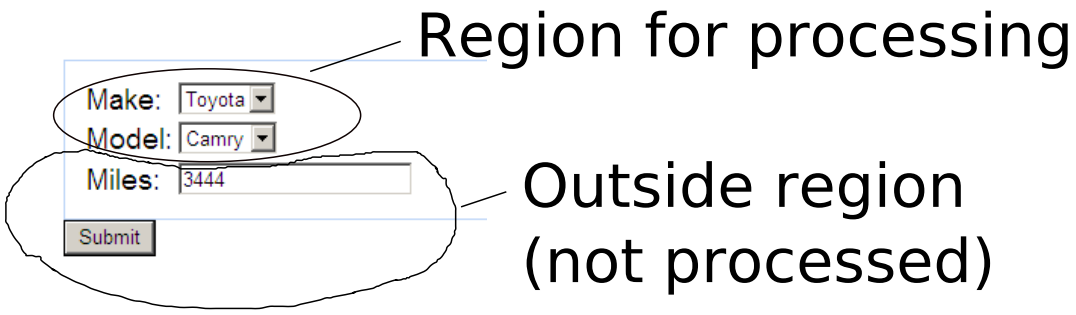
- Use Ajax and re-rendering where applicable
- Use Ajax and partial data update where applicable
- Be careful using Ajax on continuous events such as mouse-motion and keypress (use queues as necessary)
- Use standard JSF components instead of RichFaces equivalents (if you are not using the extra functionality)
- Keep logic in "get" methods to a minimum since they are called often

Ajax Re-Rendering and Partial Updates

- As we've learned in this class, you can design applications to avoid full page submission and updates
- This technique lets you reduce network bandwidth and likely increase browser update performance, too

RichFaces Regions and ajaxSingle

- **Regions** let you suppress portions of the JSF lifecycle, potentially reducing server load
- Regions are especially useful to avoid the validation phase during a partial page rendering
- You can define a single-component region using the **ajaxSingle** attribute

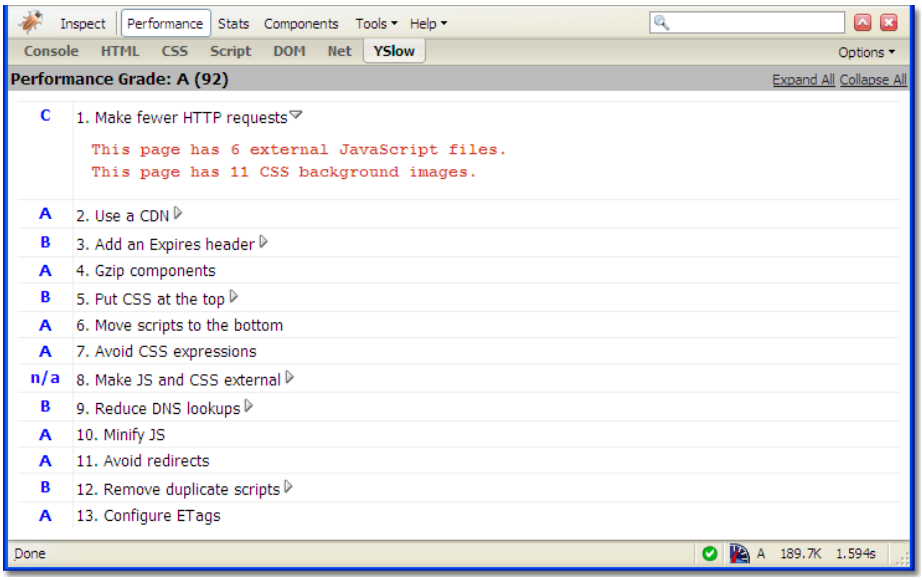


Recall that by default, the entire page is a region, so to take advantage, you must explicitly define them.

Recall that regions do not affect rendering (at least by default).

Using Firefox and Firebug

- The Firefox **Firebug** extension, in conjunction with Yahoo's **YSlow** analyzer help you identify performance issues



The Firebug home page is at:

`http://getfirebug.com/`

The YSlow home page is at:

`http://developer.yahoo.com/yslow/`

There is a nice article on using these tools to analyze Ajax preformance at:

`http://www.ibm.com/developerworks/web/library/wa-aj-perform/?ca=dgr-lnxw01FasterAjax`

Continuous Ajax Requests

- Some applications use continuous events such as mouse motion or keystrokes to trigger Ajax requests
- If you are not careful, many such requests can swamp the application and greatly reduce performance
- RichFaces provides **queue and traffic flood protection** to help guard against swamping

---

You typically use these facilities on a4j:support and related RichFaces Ajax elements.

Defining a Traffic Queue

- If you set the **eventsQueue** attribute, the RichFaces client-side runtime will queue multiple new outstanding requests if a request is already in progress
- The runtime will then send the queued requests as a batch

```
1    <h:inputText id="password"
2       value="#{queueBean.password}" >
3      <a4j:support event="onkeyup"
4        eventsQueue="myqueue"
5        reRender="length,strength" />
6    </h:inputText>
```

So in this case, if the user types a key, then quickly types several keys, if the first request is slow, all of the several key requests will be bundled into a single request. This obviously decreases network bandwidth and makes server-processing more efficient: the "queueBean.password" property in this case will be called only once for the batched keys.

Delaying Transmission

- If you set the **requestDelay** attribute, then the RichFaces client-side runtime will delay each request for the specified number of milliseconds

```
1    <h:inputText id="password"
2       value="#{queueBean.password}" >
3      <a4j:support event="onkeyup"
4        eventsQueue="myqueue"
5        requestDelay="3000"
6        reRender="length,strength" />
7    </h:inputText>
```

If you use "requestDelay" without specifying a queue, RichFaces creates a queue and derives its name from the component's ID.

Ignoring Duplicate Reponses

- If you set the **ignoreDupResponses** attribute, then the RichFaces client-side runtime will not process a response if there's an outstanding request in the queue

```
1    <h:inputText id="password"
2       value="#{queueBean.password}" >
3      <a4j:support event="onkeyup"
4        eventsQueue="myqueue"
5        ignoreDupResponses="true"
6        reRender="length,strength" />
7    </h:inputText>
```

If you use "ignoreDupResponses" without specifying a queue, RichFaces defines a queue for you, deriving the queue name from the component's ID.

Reducing the Size of Ajax Responses

- If your application will be sending a large number of Ajax requests, the size of the response (in characters) can become a bottleneck:
    - Network bandwidth usage
    - Client-side parsing

Google is famous for working very hard to reduce the number of characters sent back from its home page.

Reducing Response Size Techniques

- Here are some techniques for reducing response character size, mostly at the expense of code readability:
    - Use assigned, short component IDs
    - Use short pathnames

Some of these techniques are taken from an excellent article by Dan Allen at:

`http://www.jsfcentral.com/articles/speed_up_your_jsf_app_2.html`

Configuring RichFaces to Pack Responses

- By using the following **context-param** elements, RichFaces will combine all RichFaces CSS styles and JavaScript on the first page of an application so the browser can cache them
- This will reduce the character size of Ajax responses, especially with partial-page updates

```
1    <context-param>
2      <param-name>
3        org.richfaces.LoadScriptStrategy
4      </param-name>
5      <param-value>ALL</param-value>
6    </context-param>
7    <context-param>
8      <param-name>
9        org.richfaces.LoadStyleStrategy
10     </param-name>
11     <param-value>ALL</param-value>
12   </context-param>
```

The idea here is that when the user first goes to a page, RichFaces provides all of the scripts and styles used on that page. Then, on an Ajax request, the response header doesn't need to reference them.

If you don't use these settings, the Ajax response headers DO reference the required styles and script files, thus increasing the size of the Ajax response.

There doesn't seem to be a downside for using these settings, so it's a mystery why they aren't the default.

Configuring Facelets

- Facelets normally dynamically processes any referenced templates on each page request
- If you don't plan to change templates while the application is deployed, you can gain performance by writing a **context-param** element

```
1     <context-param>
2       <param-name>facelets.REFRESH_PERIOD</param-name>
3       <param-value>-1</param-value>
4     </context-param>
```

This will probably not make a big performance difference, but could be useful in extracting that last little bit of speed.

RichFaces Filter Configuration

- The RichFaces servlet filter "tidies up" response HTML so that Ajax responses work properly
- There are three possible configuration values:
    - **NONE** – No corrections done; markup must always be correct or odd layout issues may occur after Ajax requests
    - **TIDY (default)** – Uses the Tidy HTML parser. Slow, but robust
    - **NEKO** – Uses the NEKO parser; faster than Tidy if content is already well formed

---

The idea here is that if HTML is not well-formed, browsers will apply "corrections" before building the layout. The problem is that on Ajax requests, the browser's layout engine is not involved, so there could be a mismatch between the layout created by the browser and that expected by RichFaces JavaScript trying to update the DOM.

The RichFaces servlet therefore, by default, "corrects" response content, thus reducing "mismatch" issues.

The default HTML tidier (parser) is based on HTML Tidy. It's very robust but somewhat slow. So as an alternative, you can use the CyberNeko parser instead, which is faster if the markup is not very ill-formed.

Note that if you're using Facelets, all markup must be valid XHTML, so you may be able to get by with the NONE setting.

RichFaces Filter Configuration Example

- The following configuration indicates to use the Neko parser on all requests

```
1     <context-param>
2       <param-name>
3         org.ajax4jsf.xmlparser.ORDER
4       </param-name>
5       <param-value>NEKO</param-value>
6     </context-param>
7     <context-param>
8       <param-name>
9         org.ajax4jsf.xmlparser.NEKO
10      </param-name>
11      <param-value>.*\..*</param-value>
12    </context-param>
```

1 - 16

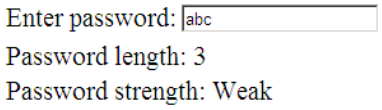
There are several other configurations possible. See the RichFaces documentation for examples.

Note that you may need to download and install the Neko parser if it's not included in your RichFaces distribution. You can download it from:

`http://sourceforge.net/projects/nekohtml/`

Complete Performance Example

- The example sends Ajax requests when the user presses keys in the input box and runs a simple password-strength test



queue.xhtml

QueueBean.java

web.xml

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2        xmlns:ui="http://java.sun.com/jsf/facelets"
3        xmlns:f="http://java.sun.com/jsf/core"
4        xmlns:h="http://java.sun.com/jsf/html"
5        xmlns:a4j="http://richfaces.org/a4j"
6        xmlns:rich="http://richfaces.org/rich">
7    <body>
8    <ui:composition template="/template.xhtml">
9        <ui:define name="title">
10           RichFaces Ajax Queue
11         </ui:define>
12       <ui:define name="body">
13           <h:form>
14              <h:panelGrid columns="2">
15                <h:outputLabel for="password"
16                  value="Enter password: " />
17                <h:inputText id="password"
18                  value="#{queueBean.password}" >
19                  <a4j:support event="onkeyup"
20                     eventsQueue="myqueue"
21                     ignoreDupResponses="true"
22                     requestDelay="3000"
23                     reRender="length,strength" />
24                </h:inputText>
25              </h:panelGrid>
26              <h:panelGrid>
27                <h:outputText id="length"
28                  value="Password length: #{queueBean.length}" />
29                <h:outputText id="strength"
30                  value="Password strength: #{queueBean.strength}" />
31              </h:panelGrid>
32           </h:form>
33           <a4j:log/>
34       </ui:define>
35   </ui:composition>
36   </body>
37   </html>
```

```
1    package mypackage;
2
3    import org.ajax4jsf.context.AjaxContext;
4
5    public class QueueBean
6    {
7        private String password;
8
9        public String getPassword()
10       {
11           return password;
12       }
13
14       public void setPassword(String password)
15       {
16           System.out.println("In setPassword: " + password);
17           this.password = password;
18
19           AjaxContext ac = AjaxContext.getCurrentInstance();
20           boolean ajax = ac.isAjaxRequest();
21           System.out.println("About to sleep, Ajax request: " + ajax);
22           try
23           {
24               Thread.sleep(10000);
25           }
26           catch (Exception e) {}
27           System.out.println("Done sleeping");
28       }
29
30       public int getLength()
31       {
32           if (password != null)
33               return password.length();
34           else
35               return 0;
36       }
37
38       public String getStrength()
39       {
40           if (password != null)
41           {
42               if (password.length() < 4)
43                   return "Weak";
44               else if (password.length() < 8)
45                   return "Moderate";
```

```
46                  else
47                      return "Strong";
48              }
49              else
50                  return "";
51          }
52      }
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3        xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/n
4        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
5        id="WebApp_ID" version="2.5">
6        <display-name>performanceWeb</display-name>
7        <welcome-file-list>
8            <welcome-file>index.html</welcome-file>
9            <welcome-file>index.htm</welcome-file>
10           <welcome-file>index.jsp</welcome-file>
11           <welcome-file>default.html</welcome-file>
12           <welcome-file>default.htm</welcome-file>
13           <welcome-file>default.jsp</welcome-file>
14       </welcome-file-list>
15       <servlet>
16           <servlet-name>Faces Servlet</servlet-name>
17           <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
18           <load-on-startup>1</load-on-startup>
19       </servlet>
20       <servlet-mapping>
21           <servlet-name>Faces Servlet</servlet-name>
22           <url-pattern>*.faces</url-pattern>
23       </servlet-mapping>
24       <context-param>
25           <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
26           <param-value>.xhtml</param-value>
27       </context-param>
28       <filter>
29           <display-name>RichFaces Filter</display-name>
30           <filter-name>richfaces</filter-name>
31           <filter-class>org.ajax4jsf.Filter</filter-class>
32       </filter>
33       <filter-mapping>
34           <filter-name>richfaces</filter-name>
35           <servlet-name>Faces Servlet</servlet-name>
36           <dispatcher>REQUEST</dispatcher>
37           <dispatcher>FORWARD</dispatcher>
38           <dispatcher>INCLUDE</dispatcher>
39       </filter-mapping>
40       <context-param>
41           <param-name>org.richfaces.SKIN</param-name>
42           <param-value>classic</param-value>
43       </context-param>
44       <context-param>
45           <param-name>facelets.BUILD_BEFORE_RESTORE</param-name>
```

```
46          <param-value>true</param-value>
47      </context-param>
48      <context-param>
49          <param-name>org.ajax4jsf.xmlparser.ORDER</param-name>
50          <param-value>NONE</param-value>
51      </context-param>
52      <context-param>
53          <param-name>org.ajax4jsf.xmlparser.NONE</param-name>
54          <param-value>.*\..*</param-value>
55      </context-param>
56      <context-param>
57          <param-name>org.richfaces.LoadScriptStrategy</param-name>
58          <param-value>ALL</param-value>
59      </context-param>
60      <context-param>
61          <param-name>org.richfaces.LoadStyleStrategy</param-name>
62          <param-value>ALL</param-value>
63      </context-param>
64      <context-param>
65          <param-name>facelets.REFRESH_PERIOD</param-name>
66          <param-value>-1</param-value>
67      </context-param>
68  </web-app>
```

Chapter Summary

In this chapter, you learned:

- Some general RichFaces design tips
- About various RichFaces performance enhancing techniques