How To Create Email From JSF Templates

By Roger Keays, 1 October 2012

Hi everybody. In this blog I'm going to share a little trick for **creating emails** using **JSF Facelets templates**. The concept is actually independent of JSF and could be used for any servlet-based technology.

Right, so the basic idea is to temporarily **trick the ServletResponse** object into writing all the content to an **in-memory buffer**, rather than streaming it to the client's browser. We then manually invoke the JSF render phase for the particular view using the special ServletResponse and voila!, the captured output can be put into an email.

Let's have a look at some of the code.

First, the heavy lifting. Here is the method which sets up the context correctly for JSF to create and render a template without messing up the current view. This method also lets you pass in a map of parameters that get put in the request scope while the template is rendered.

```
/**
 * Render a template in memory and return the content as a string. The
 * request parameter 'emailClient' is set to true during rendering. This
 * method relies on a FacesContext for Facelets templating so it only
 * works when the app is deployed.
 */
public static String capture(String template, Map params) {

    // setup a response catcher
    FacesContext faces = FacesContext.getCurrentInstance();
    ExternalContext context = faces.getExternalContext();
    ServletRequest request = (ServletRequest) faces.getExternalContext().
            getRequest();
    HttpServletResponse response = (HttpServletResponse)
            context.getResponse();
    ResponseCatcher catcher = new ResponseCatcher(response);

    // hack the request state
    UIViewRoot oldView = faces.getViewRoot();
    Map oldAttributes = null;
    if (params != null) {
        oldAttributes = new HashMap(params.size() * 2); // with buffer
```

```
            for (String key : (Set<String>) params.keySet()) {
                oldAttributes.put(key, request.getAttribute(key));
                request.setAttribute(key, params.get(key));
            }
        }
        request.setAttribute("emailClient", true);
        context.setResponse(catcher);

        try {
            // build a JSF view for the template and render it
            ViewHandler views = faces.getApplication().getViewHandler();
            UIViewRoot view = views.createView(faces, template);
            faces.setViewRoot(view);
            views.getViewDeclarationLanguage(faces, template).
                    buildView(faces, view);
            views.renderView(faces, view);
        } catch (IOException ioe) {
            String msg = "Failed to render " + template;
            faces.addMessage(null, new FacesMessage(
                    FacesMessage.SEVERITY_ERROR, msg, msg));
            return null;
        } finally {

            // restore the request state
            if (oldAttributes != null) {
                for (String key : (Set<String>) oldAttributes.keySet()) {
                    request.setAttribute(key, oldAttributes.get(key));
                }
            }
            request.setAttribute("emailClient", null);
            context.setResponse(response);
            faces.setViewRoot(oldView);
        }
        return catcher.toString();
    }
```

The important lines are shown in bold above. Now that you have your view rendered as a string you can easily turn it into an email.

The **ResponseCatcher** which actually captures the output of the rendering is a pretty dumb object. It just wraps the existing response object, but **intercepts getWriter()** method to return a local in-memory CharArrayWriter:

```
/**
 * This is a response wrapper which saves all of the output to a char array,
 * so it can be retrieved as a string afterwards with the toString() method.
 * We only support capturing text output currently.
 */
public class ResponseCatcher implements HttpServletResponse {

    /** the backing output stream for text content */
    CharArrayWriter output;

    /** a writer for the servlet to use */
    PrintWriter writer;

    /** a real response object to pass tricky methods to */
    HttpServletResponse response;

    /**
     * Create the response wrapper.
     */
    public ResponseCatcher(HttpServletResponse response) {
        this.response = response;
        output = new CharArrayWriter();
        writer = new PrintWriter(output, true);
    }

    /**
     * Return a print writer so it can be used by the servlet. The print
     * writer is used for text output.
     */
    public PrintWriter getWriter() {
        return writer;
    }

    public void flushBuffer() throws IOException {
        writer.flush();
    }

    public boolean isCommitted() {
        return false;
    }
```

```
public boolean containsHeader(String arg0) {
    return false;
}

/* wrapped methods */
public String encodeURL(String arg0) {
    return response.encodeURL(arg0);
}

public String encodeRedirectURL(String arg0) {
    return response.encodeRedirectURL(arg0);
}

public String encodeUrl(String arg0) {
    return response.encodeUrl(arg0);
}

public String encodeRedirectUrl(String arg0) {
    return response.encodeRedirectUrl(arg0);
}

public String getCharacterEncoding() {
    return response.getCharacterEncoding();
}

public String getContentType() {
    return response.getContentType();
}

public int getBufferSize() {
    return response.getBufferSize();
}

public Locale getLocale() {
    return response.getLocale();
}

public void sendError(int arg0, String arg1) throws IOException {
    response.sendError(arg0, arg1);
}
```

```
public void sendError(int arg0) throws IOException {
    response.sendError(arg0);
}

public void sendRedirect(String arg0) throws IOException {
    response.sendRedirect(arg0);
}

@Override
public void addCookie(Cookie arg0) {
    response.addCookie(arg0);
}

@Override
public void setDateHeader(String arg0, long arg1) {
    response.setDateHeader(arg0, arg1);
}

@Override
public void addDateHeader(String arg0, long arg1) {
    response.addDateHeader(arg0, arg1);
}

@Override
public void setHeader(String arg0, String arg1) {
    response.setHeader(arg0, arg1);
}

@Override
public void addHeader(String arg0, String arg1) {
    response.addHeader(arg0, arg1);
}

@Override
public void setIntHeader(String arg0, int arg1) {
    response.setIntHeader(arg0, arg1);
}

@Override
public void addIntHeader(String arg0, int arg1) {
```

```
        response.addIntHeader(arg0, arg1);
    }

    @Override
    public void setContentLength(int arg0) {
        response.setContentLength(arg0);
    }

    @Override
    public void setContentType(String arg0) {
        response.setContentType(arg0);
    }

    /* null ops */
    @Override public void setStatus(int arg0) {}
    @Override public void setStatus(int arg0, String arg1) {}
    @Override public void setBufferSize(int arg0) {}
    @Override public void resetBuffer() {}
    @Override public void reset() {}
    @Override public void setLocale(Locale arg0) {}


    /* unsupported methods */
    public ServletOutputStream getOutputStream() throws IOException {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    /**
     * Return the captured content.
     */
    @Override
    public String toString() {
        return output.toString();
    }
}
```

What about JSF 1.2?

If you're running JSF 1.2, you can still use the same ResponseCatcher but the capture() method looks a little different. I'm currently only using this with Facelets, but I used to do something similar with the JSP ViewHandler so I'd expect it to work okay with JSP too.

Here is the JSF 1.2 version:

```
/**
 * Render a view in memory and return the content as a string. The
 * request parameter 'emailClient' is set to true during rendering.
 */
public String capture(String template) {

    // initial values
    FacesContext faces = FacesContext.getCurrentInstance();
    ExternalContext context = faces.getExternalContext();
    HttpServletResponse response = (HttpServletResponse)
            context.getResponse();
    ResponseCatcher catcher = new ResponseCatcher(response);
    ViewHandler views = faces.getApplication().getViewHandler();

    // render the message
    try {
        context.setResponse(catcher);
        context.getRequestMap().put("emailClient", true);
        views.renderView(faces, views.createView(faces, template));
        context.getRequestMap().remove("emailClient");
        context.setResponse(response);
    } catch (IOException ioe) {
        String msg = "Failed to render email internally";
        faces.addMessage(null, new FacesMessage(
                FacesMessage.SEVERITY_ERROR, msg, msg));
        return null;
    }
```

That's it. Unfortunately it doesn't get much simpler (although one developer has had success replacing the JSF ResponseWriter instead of the ServletResponse). If this method doesn't work for you, head on over and try out his code.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.