## 1. XML Based Configuration

Before Servlet 3.0, we'd configure a Java web application in a web.xml file:

<**web-app** xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

version="2.5">

<**listener**>

<**listener-class**>com.baeldung.servlets3.web.listeners.RequestListener</**listener-class**>

</**listener**>

<**servlet**>

<**servlet-name**>uppercaseServlet</**servlet-name**>

<**servlet-class**>com.baeldung.servlets3.web.servlets.UppercaseServlet</**servlet-class**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>uppercaseServlet</**servlet-name**>

<**url-pattern**>/uppercase</**url-pattern**>

</**servlet-mapping**>

<**filter**>

<**filter-name**>emptyParamFilter</**filter-name**>

<**filter-class**>com.baeldung.servlets3.web.filters.EmptyParamFilter</**filter-class**>

</**filter**>

<**filter-mapping**>

<**filter-name**>emptyParamFilter</**filter-name**>

<**url-pattern**>/uppercase</**url-pattern**>

</**filter-mapping**>

</**web-app**>

Let's start replacing each configuration section with the respective annotations introduced in Servlet 3.0.

## 2. The Maven Dependency

In order to use these new annotations, we need to include the *[javax.servlet-api](https://search.maven.org/search?q=g:javax.servlet%20a:javax.servlet-api)* dependency:

<**dependency**>

<**groupId**>javax.servlet</**groupId**>

<**artifactId**>javax.servlet-api</**artifactId**>

<**version**>4.0.1</**version**>

</**dependency**>

### https://github.com/eugenp/tutorials/tree/master/spring-mvc-java

## 4. Servlets

JEE 6 shipped with Servlet 3.0 which enables us to use annotations for servlet definitions, minimizing the use of a *web.xml* file for a web application.

**For example, we can define a servlet and expose it with the *@WebServlet* annotation**

Let's define one servlet for the URL pattern */uppercase*. It will transform the value of the*input* request parameter to uppercase:

@WebServlet(urlPatterns = "/uppercase", name = "uppercaseServlet")

**public** **class** **UppercaseServlet** **extends** **HttpServlet** {

**public** **void** **doGet**(HttpServletRequest request, HttpServletResponse response)

**throws** IOException {

String inputString = request.getParameter("input").toUpperCase();

PrintWriter out = response.getWriter();

out.println(inputString);

}

**Note that we defined a name for the servlet (uppercaseServlet) that we can now reference.** We'll make use of this in the next section.

With the @WebServlet annotation, we're replacing the servlet and servlet-mapping sections from the web.xml file.

**5. Filters**

A *Filter* is an object used to intercept requests or responses, performing pre- or post-processing tasks.

**We can define a filter with the *@WebFilter* annotation.**

Let's create a filter to check if the *input* request parameter is present:

@WebFilter(urlPatterns = "/uppercase")

**public** **class** **EmptyParamFilter** **implements** **Filter** {

@Override

**public** **void** **doFilter**(ServletRequest servletRequest, ServletResponse servletResponse,

FilterChain filterChain) **throws** IOException, ServletException {

String inputString = servletRequest.getParameter("input");

**if** (inputString != **null** && inputString.matches("[A-Za-z0-9]+")) {

filterChain.doFilter(servletRequest, servletResponse);

} **else** {

servletResponse.getWriter().println("Missing input parameter");

}

}

// implementations for other methods

}

With the *@WebFilter* annotation, we're replacing the *filter* and *filter-mapping* sections from the *web.xml* file.

## 6. Listeners

We'll often need to trigger actions based on certain events. This is where listeners come to the rescue. These objects will listen for an event and execute the behavior we specify.

**Like previously, we can define a listener with the @WebListener annotation.**

Let's create a listener that counts each time we perform a request to the server. We'll implement ServletRequestListener, listening for ServletRequestEvents:

@WebListener

**public** **class** **RequestListener** **implements** **ServletRequestListener** {

@Override

**public** **void** **requestDestroyed**(ServletRequestEvent event) {

HttpServletRequest request = (HttpServletRequest)event.getServletRequest();

**if** (!request.getServletPath().equals("/counter")) {

ServletContext context = event.getServletContext();

context.setAttribute("counter", (**int**) context.getAttribute("counter") + 1);

}

}

// implementations for other methods

}

Note that we are excluding the requests to the URL pattern /counter.

With the @WebListener annotation, we're replacing the listener section from the web.xml file.