Scripting and JavaEE 6 alignment

Servlet 3.0 PRD Section 8 describes pluggability. As part of this pluggability, Servlet 3.0 JSR defines modular web.xml. The following web frameworks can be composed with a JavaEE web application:

Co-hosting a Rails and Servlet in a web application

Servlet 3.0 is proposing pluggability mechanism where it will make it easy for frameworks (including scripting) to be packaged along with JavaEE applications. For example packaging a Rails and Servlet/JSP together.

It is expected that, to deploy a Rails application on a Servlet 3.0 compliant container, you would need to simply place the
Rails application inside WEB-INF.

The Jruby runtime jar and jruby-rack.jar can either be packaged along with the application (WEB-INF/lib) or can be made available by the container. For example, GlassFish can provide jruby-rack.jar as one of it's module or it can simply be copied on Tomcat inside $TOMCAT_HOME/shared/lib.

JRuby-Rack can take advantage of Servlet 3.0 in the following ways:

- Provide implementation of ServletContainerInitializer interface. Following goals will be achieved with it
- web.xml will not be needed
- Implementation of ServletContainerInitializer will detect if the appplication in-scope is a Rails application and do all the required initializations, such as, JRuby runtime creation among other things. Refer to ServletContext new APIs in Servlet 3.0 PRD.

Here is the latest proposed ServletContainerInitializer:

```
package javax.servlet;
ServletContainerInitializer {
  public abstract void onStartup(Set<Class<?>>, ServletContext ctx);
}
```

An instance of this is looked up via the jar services API. The framework providing an implementation of this Initializer would bundle in the META-INF/services directory of the jar file a file called javax.servlet.WebContainerInitializer that points to the implementation class.

```
RailsContainerInitializer extends ServletContainerInitializer {
  public void onStartup(Set<Class <?>>c , ServletContext ctx) {
    //c will be null if ServletContainerInitializer does not have a @HandlesTypes annotation on it.

    InputStream boot = ctx.getResourceAsStream("WEB-INF/environment/boot.rb");

   //There is a Rails application inside this web archive, lets add the servlet/filter etc.
    if(boot != null){
        FilterRegistration fr = ctx.addFilter(“JRuby Rack Filter”, flterClass);
        //Use the FilterRegistration fr to add the Filter related metadata.
    }
  }
}
```

Issues identified so far

- How to map requests to static content in Rails apps to WEB-INF/public.

  JRuby-rack is aware of WEB-INF/public. For the static contents it can always detect if the request is for static content and serve these using ServletContext.getResourceAsStream(String) API.

- How to have the files in WEB-INF/public handled by the container's default file handling

  Either using container's alternate docroot or by simply copying contents of WEB-INF/public to the root
  of the WAR.

- Does the JRuby or Rails runtime depend on being able to access the contents of the war file as **files**? The spec doesn't require that a war file be unpacked and available as individual files in the filesystem. It would be best if the runtime used the Servlet container's resource access methods instead of java.io.File APIs.

  Files loaded by a Rails application code using load or require can be handled using ServletContext.getResource()/getResourceAsStream(). However, if the Rails application code uses ruby's File API to load a ruby file, it will not work.

- JRuby-Rack responds to few servlet context init parameters.

  It can still do so using reasonable defaults or alternate way of configuring it that doesn't depend on web.xml/context init parameters. For example, it can add the context init prams from the ServletContext that is passed in to the ServletContainerInitializer. There is a method there that allows adding context init params and init params.

Groovy GSP and Java Servlet

GSP means GroovyServer Pages, which is similar to JSP (JavaServer Pages) and written in groovy. GSP is enabled through a Servlet GroovyPages. Following is the web.xml that can be used with other Servlet(s) to process GSP files:

```
<web-app>
     <servlet>
         <servlet-name>GSP</servlet-name>
         <servlet-class>groovy.modules.pages.GroovyPages</servlet-class>
         <init-param>
              <param-name>encoding</param-name>
              <param-value>ISO-8859-1</param-value>
         </init-param>
         <init-param>
              <param-name>debug</param-name>
              <param-value>0</param-value>
         </init-param>
     </servlet>

     <servlet-mapping>
         <servlet-name>GSP</servlet-name>
         <url-pattern>*.gsp</url-pattern>
     </servlet-mapping>
     </servlet>
```

```
</web-app>
```

The GSP servlet jar can contain a web-fragment (META-INF/web-fragment.xml) such that any other servlet applicatin that wants to add support for GSP don't need to incorporate the above web.xml, all they need to do is to place the jar containing GSP Servlet inside WEB-INF/lib directory. The main web.xml or application web.xml should not have metadata-complete='true' otherwise the web-fragment included inside the GSP Servlet will be ignored. Refer to Servlet 3.0 section 8 for more details. Here is the web-fragment.xml for the GSP jar:

```
<web-fragment>
     <servlet>
         <servlet-name>GSP</servlet-name>
         <servlet-class>groovy.modules.pages.GroovyPages</servlet-class>
         <init-param>
              <param-name>encoding</param-name>
              <param-value>ISO-8859-1</param-value>
         </init-param>
         <init-param>
              <param-name>debug</param-name>
              <param-value>0</param-value>
         </init-param>
     </servlet>

     <servlet-mapping>
         <servlet-name>GSP</servlet-name>
         <url-pattern>*.gsp</url-pattern>
     </servlet-mapping>
     </servlet>
</web-fragment>
```

GSP project started off at http://gsp.dev.java.net and now moved to http://grails.org.