Java Servlets & ServerPages / JDBC / Sybase ....on Linux !

This document outlines how to setup:

- the Blackdown JDK
- Sun's JSDK Java Servlet Development Kit
- a Java Servlet environment on a Linux machine with Apache-JServ.
- a Java Server Pages (JSP) configuration.
- a Sybase database on Linux.
- a Sybase JDBC Driver.

Whew. That's a lot, so lets get going..

Blackdown JDK

I used to use the RPMS for the JDK, I don't anymore. They are a piece of cake to install by hand, and I have run into two instances where the JDK RPM is simply broken :(

Download the tarball from a blackdown mirror the site is http://www.blackdown.org/
I currently have this one installed. jdk_1_1_7-v1a-glibc-x86.tar.gz

To install :
download this package to a `/directory/somplace/`
and as *`root`*

```
cd /usr/lib
tar -xzvf /directory/someplace/jdk_1_1_7-v1a-glibc-x86.tar.gz
ln -s jdk117_v1a/ jdk-1.1.7
```

I then wrote a quick script to emulate the envoronment settings that the rpm usually sets up.
The file is:
`/usr/local/bin/java.env`

```
##      java.env        ##

## set environment variables for the Java(tm) JDK
## that would otherwise be set by the RPM package
JDK_HOME=/usr/lib/jdk-1.1.7
JAVA_HOME=/usr/lib/jdk-1.1.7
CLASSPATH=/usr/lib/jdk-1.1.7/lib/classes.zip:.:$CLASSPATH
PATH=/usr/lib/jdk-1.1.7/bin:$PATH

export JDK_HOME JAVA_HOME CLASSPATH PATH

##      end java.env    ##
```

I then put a line in the `/etc/profile:`

```
source /usr/local/bin/java.env
```

That's it. I like to logout and log in again to make sure everything is setup ok. Before we get into logging out, we can setup the

Java Servlet Development Kit.

The file you want is: `jsdk20-solaris2-sparc.tar`

You can download it from www.sun.com, or www.javasoft.com

Installation is just like the jdk, we will go to the directory `/usr/lib` and do a

```
tar -xvf /directory/somewhere/jsdk20-solaris2-sparc.tar
```

```
then do a
```

```
ln -s JSDK2.0/  jdsk
```

Now, here is the reason we did not logout and log in again, we are going to change the `/usr/local/bin/java.env` file one more time.

on the `CLASSPATH` line, we make a change:

```
CLASSPATH=/usr/lib/jdk-
1.1.7/lib/classes.zip:/usr/lib/jsdk/lib/jsdk.jar:.:$CLASSPATH
```

This adds the Servlet base classes to the CLASSPATH so they will be found.

Ok, log out, log in as root.

Now let's test our conection :)

as root, go to the directory `/usr/lib/jsdk/bin`
you will find a program there called servletrunner. we will invoke it in (-v) verbose mode.

```
./servletrunner -v
servletrunner starting with settings:
  port = 8080
  backlog = 50
  max handlers = 100
  timeout = 5000
  servlet dir = ./examples
  document dir = ./examples
  servlet propfile = ./examples/servlet.properties
```

Fire up Netscape, or another brower, and point it at this URL

```
 http://your.servername.com:8080/servlet/SimpleServlet
```

....you should see the output from the servlet. Even if you don't have a webserver running, the servletrunnner utility wil be able to listen and respond requests. If it doesn't respond, then check in the /usr/lib/jsdk/examples directory. You may need to compile the examples with:

```
javac *.java
```

Cool, the JDK, and JSDK are setup. Now let's move into a little more hairy territory:

Apache & Apache-JServ

First, I installed Apache-1.3.4 from source. The DSO configuration for the rpm install on RedHat has some funky configuration that confuses Apache-JServ into thinking that apache is not a DSO install...when in fact it is a DSO install.

Anyway, here is how I installed it.

I downloaded apache-1.3.4.tar.gz from www.apache.org, saved it in /usr/src. I unpacked it with

```
tar -xzvf apache-1.3.4.tar-gz.
```

then I changed directories

```
cd apache_1.3.4
```

and and ran

```
./configure --enable-module=so --enable-shared=max
```

and

```
make; make install
```

This installs apache with the proper modules. Now on to Apache-JServ....

I downloaded Apache-JServ-1.0b3.tar.gz from http://java.apache.org/ and saved it in `/usr/src/Apache-JServ-1.0b3.tar.gz`

Then I went to the main directory where apache is installed `(/usr/src/apache).` This is to keep all the configuration files together. So I...

```
cd /usr/local/apache/
```

and I unpacked the source for Apache-JServ

```
tar -xzvf /usr/src/Apache-JServ-1.0b3.tar.gz
```

Then I moved the directory name so that it doesn't bother with the version number.

```
mv Apache-JServ-1.0b3/ Apache-JServ
```

and we go into the Apache-JServ directory

```
cd Apache-JServ/
```

Now we confgure Apache-JServ into the existing install of Apache:

We are in the /usr/local/apache/Apache-JServ directory, so we issue the command:

```
./configure --with-jdk-home=/usr/lib/jdk-1.1.7/ --with-
jsdk=/usr/lib/jsdk/lib/jsdk.jar --with-apache-install=/usr/local/apache/
```

You will likely see some messages like this:

```
Apache Directory: /usr/local/apache/
Apache Version: 1.3
Module Type: dynamic (DSO will be used to link mod_jserv into server dynamically)
Apache include flags: -I/usr/local/apache/include


Run make to build jserv.

Then, to run the example put this in your httpd.conf:

Include /usr/local/apache/Apache-JServ/example/jserv.conf

Then try visiting the URL:

http://wakame.moshi.com:SERVER_PORT/example/Hello

If that works then you have successfully setup Apache JServ.
You might consider putting frequently used options into ./configure-options

For further information please read the documentation.

----------------------------------
```

[This information above is very useful, it may be a good idea to save the text in this window to a file for reference.] We have configured Apache-JServ to use the current install of the Apache web server, now let's install it:

```
make; make install
```

After the command prompt returns, we'll configure the current Apache install to use Apache-JServ.

```
cd /usr/local/apache/conf/
```

Then we'll copy the configuration file from the Apache-JServ directory, and include it with the rest of the configuration files.

```
cp /usr/local/apache/Apache-JServ/example/jserv.conf .
```

Then, rather than open the file and type in the configuration, we'll just append the one line we need to the file directly :)

```
 echo "Include /usr/local/apache/conf/jserv.conf " >> httpd.conf
```

Now we should be able to start up apache and see that the Apache-JServ stuff is working. Here is the command:

```
/usr/local/apache/bin/apachectl start
```

Now, you should be able to fire up your browser and see the sample page.

```
http://localhost/example/Hello
```

After I got it working I cleaned up the configuration so that all the configuration files were in one spot. I also included the configuration necessary to get JavaServerPages configured.
I modified the `/usr/local/apache/conf/jserv.conf` file like this:

```
###############################################################################
#                     Apache JServ Configuration File                         #
###############################################################################
.
.
```

```
.
#ApJServProperties /usr/local/apache/Apache-JServ/example/jserv.properties
ApJServProperties /usr/local/apache/conf/jserv.properties
.
.

#ApJServLogFile /usr/local/apache/Apache-JServ/example/jserv.log
ApJServLogFile /usr/local/apache/logs/jserv.log
.
.
.
# Example: "ApJServMount /servlets ajpv11://jserv.mydomain.com:15643/myServlets"
#  if user requests "http://host/servlets/TestServlet" the servlet
#  "TestServlet" in zone "myServlets" on host "jserv.mydomain.com" using
#  "ajpv11" protocol on port "15643" will be executed
ApJServMount /example /example
ApJServMount /servlets /live
.
.
# Notes: This is used for external tools such as JSP (Java Servlet Pages),
#  GSP (GNU Server Pages) or Java server side include.
#ApJServAction .jsp /servlets/nl.nmg.jsp.JSPServlet
ApJServAction .jsp /servlets/org.gjt.jsp.JSPServlet
.
.
###############################################################################
```

The bold lines are the changes I made to the file. The GnuJSP line will come later, note that merely uncommenting the line gives an old--and wrong--configuration. Now let's change the file: **`/usr/local/apache/conf/jserv.properties`**

```
###################################
# Apache JServ Configuration File #
###################################
.
.
# CLASSPATH environment value passed to the JVM
# Syntax: wrapper.classpath=[path]
# Default: NONE (Sun's JDK/JRE already have a default classpath)
# Notes: if more than one line is supplied these will be concatenated using
#        ":" or ";" (depending wether Unix or Win32) characters.
#        JVM must be able to find JSDK and JServ classes and any
#        utility classes used by your servlets.
wrapper.classpath=/usr/lib/jdk-1.1.7//lib/classes.zip
#wrapper.classpath=/usr/local/apache/Apache-JServ/src/java/Apache-JServ.jar
wrapper.classpath=/usr/local/apache/jarfiles/Apache-JServ.jar
wrapper.classpath=/usr/local/apache/jarfiles/gnujsp.jar
wrapper.classpath=/usr/lib/jsdk/lib/jsdk.jar
.
.
###########################
# Servlet Zones parameters
###########################

# List of servlet zones JServ manages
# Syntax: zones=<servlet zone>,<servlet zone>... (Comma separated list of String)
# Default: <empty>
zones=example,live
# Configuration file for each servlet zone
# Syntax: <servlet zone name as on the zones list>.properties=<full path to
configFile> (String)
# Default: <empty>
example.properties=/usr/local/apache/conf/example.properties
live.properties=/usr/local/apache/conf/live.properties
```

.
.

You'll probably notice that I setup another "servlet zone" called "live". To do this I copied the example.properties file from `/usr/local/apache/Apache-JServ/example/` to `/usr/local/apache/conf`. Then I went into `/usr/local/apache/conf` and copied `example.properties` to `live.properties`.

I then edited `live.properties`, this is the file that the JSP pages will need to use. Here are the changes I made to `live.properties.`

```
###############################################################################
#                        Servlet Zone Configuration File                      #
###############################################################################
.
.
# List of Repositories
#######################

# The list of servlet repositories controlled by this servlet zone
# Syntax: repositories=[repository],[repository]...
# Default: NONE
repositories=/usr/local/apache/Apache-JServ/live
.
.
# Startup Servlets
###################

# Comma or space delimited list of servlets to launch on startup.
# This can either be a class name or alias.
# Syntax: servlets.startup=[classname or alias],[classname or alias],...
# Default: NONE
# servlets.startup=hello,snoop,org.fool.Dummy
servlets.startup=org.gjt.jsp.JSPServlet
.
.
# Servlet Init Parameters
##########################

# These properties define init parameters for each servlet that is invoked
# by its classname.
# Syntax: servlet.[classname].initArgs=[name]=[value],[name]=[value],...
# Default: NONE
# servlet.org.fool.Dummy.initArgs=message=I'm a dummy servlet
servlet.org.gjt.jsp.JSPServlet.initArgs=repository=/usr/local/apache/gnujspclasses
.
.
.
```

Note: the directory `/usr/local/apache/gnujspclasses` needs to be created and needs to have the permissions set so that it is world writable. This is the directory where the JSP source code is generated, the classes compiled, and the packages are built. No write--no work :)

Apache-JServ-SSI

SSI stands for Sever Side Include and it enables stuff like
<SERVLET CODE=HelloDork> </SERVLET>
To be embedded in your code so you can use templates and such. This used to be included in the Apache-JServ development tree but it has been pruned off recently to keep the JServ development from getting bogged down.

The README instructs you how to make the package containing the SSI classes. The javac compile might need to be tried twice, the second time it worked fine. I think it might be the package creation that bombs the first time. Then copy the jar file to

```
/usr/local/apache/jarfiles/ApacheJServSSI.jar
```

I had to make the `jarfiles` subdiretory first, but that shouldn't be hard.

The hardest part was to figure out where the configuration arguments go. There are a couple of pieces to note.
the "wrapper.classpath" needs to point to the "jar file" package containg the SSI stuff.
add a line like this in the jserv.properties file

```
wrapper.classpath=/usr/local/apache/jarfiles/ApacheJServSSI.jar
```

The real stinker was to note that the path name in the commented "jhtml" line is wrong.
note the directory structure in the "src" dir when you unpacked the Apache-JServ-SSI stuff.
You will need to change the line

```
#ApJServAction .jhtml /servlets/org.apache.ssi.SSIServlet
```

to read

```
ApJServAction .jhtml /servlets/org.apache.jservssi.JServSSI
```

GnuJSP

It wasn't until I started using the server-side includes that I realized that this was not going to give me the flexibility I needed. Then I perused through the O'Reilly book *Java Servlet Programming*--easliy the best Servlet book out there--and found some stuff on Java Server Pages. They are similar to PHP or ASP pages, but they dynamically generate Java Servlets upon execution, and reuse them from cache. This was what I needed to complete the picture. The advantage to Java Server Pages was that I did not have to maintain a CGI library, and I did not have to construct the web pages all by myself. Another person can write the pages and I can edit the files after the fact by simply embedding the Java code between "<% %>" tags.

I downloaded the GnuJSP pages from `http://gnujsp.carroll.com/` 0.9.8 was the current version when I configured my machine.
There is also a very helpful page that got me going, it is listed in the INSTALL document in the gnujsp distribution `http://www.bestiary.com/moose/jsp/install.html` It also contains a quick little sample file to test your jsp connections.

Installing is not really that big of a deal, configuration takes a little effort though. There is a bit of non-intuitive configuration that you did earlier. Notably the line in jserv.properties

```
ApJServMount /servlets /live
```

As the doc explains, it tells Apache to mount the servlet zone "live" as the zone "servlets."

This enables the line:

```
ApJServAction .jsp /servlets/org.gjt.jsp.JSPServlet
```

to use the "servlets" zone to build and run the Servlets for the Java Server Pages.

Confusing.... I thought so. But it doesn't take long to get the hang of it.

To test, edit a file `test.jsp` (or whatever, so long as it ends in "jsp") and put this in it:

```
<HTML>
<HEAD>
<TITLE>Testing....</TITLE>
</HEAD>
<%
for (int i=0; i<100; i++ ) {
out.println(i+"\n"); }
%>

</HTML>
```

...save it in `/usr/local/apache/htdocs/` and restart apache.
(`/usr/local/apache/bin/apachectl restart`)

Then go to http://localhost/test.jsp
you should see the numbers 0 thru 99.

Sybase

This is installed with two rpms that are found on the RedHat 5.2 Applications CD.

They are sybase-ase-11.0.3.3-2.i386.rpm and sybase-doc-11.0.3.3-2.i386.rpm

This will install the Sybase SQL Server and it's documentation. It will also create a user sybase. There is a great document that tells you how to get all your devices setup. The docs are quite well done--and **long**-- so I won'r duplicate that fine effort.

All that's really necessary on the server is to place the jar file that contains the JDBC driver in the the classpath. This should be done by placing it in a handy directory. I like: `/usr/lib/jdbc/lib/sybjdbc.jar`

Then this needs to be added to the classpath. This can be done by editing the `java.env` file from earlier. To add to the Java Sever stuff, simply add a line to `jserv.properties` file like the others that point to the jar file

```
wrapper.classpath=/usr/lib/jsdk/lib/jsdk.jar
wrapper.classpath=/usr/lib/jdbc/lib/sybjdbc.jar
```

I'm not sure about the name of the jdbc file, this is the one I have, there are other Sybase JDBC drivers from different vendors.

I can include more examples of the jsp code that uses a JDBC connnection, but I'm going to go have a beer now.