Jython webapp for DSpace

Basic Jython webapp

1. Create the webapp directory (you may use any name you want):
   `mkdir -p [dspace]/webapps-jython/WEB-INF/lib`
   Tip: The jython webapp is just another webapp like the individual DSpace webapps. So while you could put it next to the DSpace webapps into `[dspace]/webapps/jython/`, it's preferable to choose a different location (e.g. `[dspace]/webapps-jython/`) because the `[dspace]/webapps/` directory is replaced every time you run "`ant update`" (the old webapps directory will not be deleted, it will be renamed to "webapps-[timestamp]").
2. Download the latest Jython installer jar (e.g. `jython-installer-2.7.1.jar`) from ~~http://www.jython.org/downloads.html~~ (the jython.org website was last updated around 2015 [issue2658]; check Maven Central for latest jython version)
   `curl -O -J http://search.maven.org/remotecontent?filepath=org/python/jython-installer/2.7.1/jython-installer-2.7.1.jar`
3. Get `jython.jar` and the `Lib` directory.;
   a. either unzip the installer jar:
      `unzip -d [dspace]/lib/ jython-installer-2.7.1.jar jython.jar 'Lib/*'`
      `unzip -d [dspace]/webapps-jython/WEB-INF/lib/ jython-installer-2.7.1.jar jython.jar 'Lib/*'`
   b. or use it to install Jython:
      `java -jar jython-installer-2.7.1.jar --console`
      Note: Installation location doesn't matter, this is not necessary for DSpace. You can safely delete it after you retrieve `jython.jar` and `Lib`
4. Associate .py files with Jython's PyServlet

[dspace]/webapps/jython/WEB-INF/web.xml

```
<web-app>
    <servlet>
        <servlet-name>PyServlet</servlet-name>
        <servlet-class>org.python.util.PyServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>PyServlet</servlet-name>
        <url-pattern>*.py</url-pattern>
    </servlet-mapping>
</web-app>
```

5. Create a Hello World servlet:

[dspace]/webapps/jython/hello.py

```
# -*- coding: utf-8 -*-
from javax.servlet.http import HttpServlet


class hello(HttpServlet):
    def doGet(self, request, response):
        self.doPost(request, response)

    def doPost(self, request, response):
        response.setContentType("text/html")
        response.setCharacterEncoding("utf-8")
        toClient = response.getWriter()
        toClient.println("<h1>Hello World!</h1>")
        toClient.println(u"<p>To make sure that utf-8 works, here's a Czech pangram for you:</p><p>Příliš
žluouký k úpl ábelské ódy.</p>")
```

Access to DSpace classes from Jython

6. Copy DSpace jars to the jython webapp's lib directory:
   `cp -r [dspace]/lib/* [dspace]/webapps-jython/WEB-INF/lib/`
7. Start up DSpace kernel on webapp startup and point it to your DSpace configuration:

[dspace]/webapps-jython/WEB-INF/web.xml

```
<web-app>
    ...
    <!-- DSpace Configuration Information -->
    <context-param>
        <param-name>dspace-config</param-name>
        <param-value>/dspace/config/dspace.cfg</param-value>
    </context-param>
    <!-- new ConfigurationService initialization for dspace.dir -->
    <context-param>
        <description>The location of the main DSpace configuration file</description>
        <param-name>dspace.dir</param-name>
        <param-value>/dspace</param-value>
    </context-param>
    <listener>
        <listener-class>org.dspace.app.util.DSpaceContextListener</listener-class>
    </listener>
    <listener>
        <listener-class>org.dspace.servicemanager.servlet.DSpaceKernelServletContextListener</listener-
class>
    </listener>
</web-app>
```

8. Define the context in Tomcat's configuration. There are several ways how you can do that, so just use the same way you use for configuring DSpace contexts. The recommended one is to use a context fragment:

sudo vim /etc/tomcat8/Catalina/localhost/jython.xml

```
<Context docBase="/dspace/webapps-jython" reloadable="true" />
```

A few seconds after you save the file, Tomcat will notice it and load the "jython" context.

Adding Java libraries

1. Copy the .jar to `/dspace/webapps-jython/WEB-INF/lib/`
2. Reload the context
   `sudo touch /etc/tomcat8/Catalina/localhost/jython.xml`

Tip: If you forgot which libraries you added (because it's hard to spot them among dozens of libraries which belong to DSpace), here's how you can filter out the DSpace libraries, which should leave you only with a list of libraries you added:

```
diff -u <(ls -1 /dspace/webapps-jython/WEB-INF/lib/) <(ls -1 /dspace/lib/) | view -
```

Adding Python libraries

Python libraries can either be added to `/dspace/webapps-jython/WEB-INF/lib/Lib/` or to context root (`/dspace/webapps-jython/`).

Installing pure-python modules via pip: refer to https://github.com/openhab-scripters/openhab-helper-libraries/wiki/Using-PIP-in-Docker

Read the entire page including notes and warnings, it describes issues you will run into.

```
# install pip (do not upgrade pip after installing it):
JYTHON_HOME=[dspace]/webapps-jython/WEB-INF/lib/ java -jar [dspace]/webapps-jython/WEB-INF/lib/jython.jar -m
ensurepip
# install a PyPI package (e.g. requests):
JYTHON_HOME=[dspace]/webapps-jython/WEB-INF/lib/ java -jar [dspace]/webapps-jython/WEB-INF/lib/jython.jar -m
pip install requests
```

Creating nice action URLs

You may find a snippet like the one below to set up URL mapping for a Jython servlet. Unfortunately, it's not implemented in PyServlet, as it is more a demo than something to use in production (see this thread). For production deployment, Modjy is recommended.

web.xml

```
<web-app>
    ...
    <servlet-mapping>
        <servlet-name>SherpaRomeo</servlet-name>
        <url-pattern>/SherpaRomeo</url-pattern>
    </servlet-mapping>
</web-app>
```

Example of connection to the DSpace DB via ZxJDBC

An artificial example demonstrating a few techniques that are now possible thanks to the above:

- You can use zxJDBC, a pythonic (DB API 2.0) interface allowing the use of databases accessible via JDBC. The driver (postgresql-*.jar or ojdbc6.jar) used here is available in classpath because we copied it from /dspace/lib/.
- You can use Java libraries, demonstrated here by java.util.Properties used to read dspace.cfg.
- Here we read the database driver, connection string, user and password from dspace.cfg and then pass it to zxJDBC to create a connection.
- We could use DB API 2.0 methods like cursor.fetchall() to get query results. Here I chose to use the Python zip() function to return the query results in a custom format.
- You can use a context manager (Python "with" keyword) around a zxJDBC cursor to manage the scope of the DB transaction.
- You can't use the Python "with" keyword around the java.util.Properties as it is a Java class which doesn't implement a Python context manager.
- DB connection here is open in init() and closed in destroy() only to demonstrate the servlet's constructor and destructor. You should not keep a DB connection open for the whole time the servlet is loaded.

[dspace]/webapps-jython/db_example.py

```
# -*- coding: utf-8 -*-

from javax.servlet.http import HttpServlet
from com.ziclix.python.sql import zxJDBC
from java.util import Properties

DSPACE_DIR = '/dspace'


class db_example(HttpServlet):
    def doGet(self, request, response):
        self.doPost(request, response)

    def doPost(self, request, response):
        response.setContentType("text/html")
        response.setCharacterEncoding("utf-8")
        toClient = response.getWriter()
        toClient.println("<h1>Example of connection to the DSpace DB via ZxJDBC</h1>")

        rows = self.get_data_from_db()
        toClient.println("<h2>Results</h2>")
        toClient.println("<table>")

        toClient.println("<tr>")
        for column in rows[0]:
            toClient.println("<th>%s</th>" % column)
        toClient.println("</tr>")

        for row in rows:
            toClient.println("<tr>")
            for column in row:
                toClient.println("<td>%s</td>" % row[column])
            toClient.println("</tr>")

        toClient.println("</table>")

    def read_dspace_config(self, filename):
```

```
        """read dspace.cfg"""
        with open(filename, 'r') as f:
            props = Properties()
            props.load(f)
            return props

    def connect_db(self):
        """
        get DB config from DSpace config, connect in autocommit mode (each
        individual query is commited automatically)
        """
        self.conn = zxJDBC.connect(
            self.props.getProperty('db.url'),
            self.props.getProperty('db.username'),
            self.props.getProperty('db.password'),
            self.props.getProperty('db.driver'),
        )
        self.conn.autocommit = True

    def init(self, config):
        """servlet startup"""
        try:
            self.props = self.read_dspace_config(DSPACE_DIR + '/config/local.cfg')
        except IOError:
            self.props = self.read_dspace_config(DSPACE_DIR + '/config/dspace.cfg')
        self.connect_db()

    def destroy(self):
        """servlet shutdown: clean up DB connections"""
        self.conn.close()

    def get_data_from_db(self):
        """
        Query the DB and return a list of rows
        where each row is a dict of column names and values
        """
        with self.conn.cursor() as c:
            c.execute("SELECT version, description FROM schema_version ORDER BY version DESC")

            columns = [col[0] for col in c.description]
            rows = []
            for row in c:
                rowdata = dict(zip(columns, row))
                rows.append(rowdata)
            return rows
```

Error handling

I'm not yet sure why, but some exceptions are neither logged to tomcat, nor to dspace.log, nor to the output HTML.

Example: As mentioned above, the python requests library fails to fetch certain pages with `error: java.util.zip.DataFormatException: invalid stored block lengths`

Here's a workaround showing how to either log the error or print it in the browser.

[dspace]/webapps-jython/err_example.py

```
# -*- coding: utf-8 -*-

from javax.servlet.http import HttpServlet
import requests


class err_example(HttpServlet):
    def doGet(self, request, response):
        response.setContentType("text/plain")
        toClient = response.getWriter()
        toClient.println("BEGIN")
        try:
            r = requests.get('https://demo.dspace.org/xmlui/')  # errors out on certain gzipped pages; see
https://github.com/madler/zlib/issues/82
        except:
            import sys, traceback
            traceback.print_exc(file=sys.stdout)  # log to catalina.out
            traceback.print_exc(file=toClient)  # log to HTML output
        toClient.println("END")
```

See also

- Curation tasks in Jython