JythonUberspect

NOTES:

- this is a work-in-progress.
- there are some related code samples at the bottom of the page (below the jythonuberspect code)

UPDATES:

- added support for integers in the get method. (JRB)
- incorporated changes suggested by Kent Johnson

```
/*
 * Copyright 2000-2004 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License")
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.velocity.tools.generic.introspection;

import java.util.HashMap;
import java.util.Iterator;
import java.util.NoSuchElementException;

import org.apache.log4j.Logger;

import org.apache.velocity.util.introspection.*;

import org.python.core.*;

/**
 * a subclass of UberspectImpl that supports dynamic Jython objects.  Methods and
 * attributes can therefore be called on a jython object within a velocity template
 * without requiring additional tools (and without resorting to direct method calls
 * against the jython methods (i.e. no __findattr__ in your template)
 * @author <a href="mailto:jasonrbriggs@gmail.com">Jason R Briggs</a>
 */
public class JythonUberspect extends UberspectImpl
{
    private static final Logger log = Logger.getLogger(JythonUberspect.class.getName());

   /**
    * support the standard jython iterators (otherwise pass up to UberspectImpl)
    * in a velocity #foreach
    */
    public Iterator getIterator(Object obj, Info i) throws Exception
    {
        if (obj instanceof PySequence)
        {
            return new PySequenceIterator((PySequence)obj);
        }
        else if (obj instanceof PyDictionary)
        {
            return new PySequenceIterator(((PyDictionary)obj).values());
        }
        else
        {
            return super.getIterator(obj, i);
        }
```

```
    }

   /**
    * get the method for a jython object (or pass up)
    */
    public VelMethod getMethod(Object obj, String methodName, Object[] args, Info i) throws Exception
    {
        if (obj instanceof PyObject)
        {
            return new PyMethod(methodName);
        }
        else
        {
            return super.getMethod(obj, methodName, args, i);
        }
    }

   /**
    * get a property from a jython object for data retrieval
    */
    public VelPropertyGet getPropertyGet(Object obj, String identifier, Info i) throws Exception
    {
        if (obj instanceof PyObject)
        {
            return new PyGetProperty(identifier);
        }
        else
        {
            return super.getPropertyGet(obj, identifier, i);
        }
    }

   /**
    * get a property from a jython object for data modification
    */
    public VelPropertySet getPropertySet(Object obj, String identifier, Object arg, Info info) throws Exception
    {
        if (obj instanceof PyObject)
        {
            return new PySetProperty(identifier);
        }
        else
        {
            return super.getPropertySet(obj, identifier, arg, info);
        }
    }
}

/**
 * a jython velocity method
 */
class PyMethod implements VelMethod
{
    private static final Logger log = Logger.getLogger(JythonUberspect.class.getName());
    final PyString methodname;

    public PyMethod(String methodname)
    {
        this.methodname = new PyString(methodname);
    }

   /**
    * returns the jython method name
    */
    public String getMethodName()
    {
        return methodname.toString();
    }

   /**
    * the return type of the invoked method.  Just being lazy here and returning 'Object'
```

```
     * for everything at the moment
     */
     public Class getReturnType()
     {
         return Object.class;
     }

    /**
     * execute the jython method
     */
     public Object invoke(Object o, Object[] params)
     {
         PyObject po = (PyObject)o;
         PyObject rtn = null;
         try
         {
             // find the method attr on the python object
             PyObject meth = po.__findattr__(methodname);
             if (params == null || params.length < 1)
             {
                 rtn = meth.__call__();
             }
             else
             {
                 // build a python params array
                 PyObject[] pparams = new PyObject[params.length];
                 for (int i = 0; i < pparams.length; i++)
                 {
                     if (params[i] instanceof String)
                     {
                         pparams[i] = new PyString((String)params[i]);
                     }
                     else if (params[i] instanceof PyObject)
                     {
                         pparams[i] = (PyObject)params[i];
                     }
                     else if (params[i] instanceof Integer)
                     {
                         pparams[i] = new PyInteger(((Integer)params[i]).intValue());
                     }
                     else
                     {
                         System.err.println("unsupported param type " + params[i].getClass().getName());
                         log.error("unsupported param type : " + params[i].getClass().getName());
                         return null;
                     }
                 }

                 rtn = meth.__call__(pparams);
                 if (rtn instanceof PyNone)
                 {
                     rtn = null;
                 }
             }

             return rtn;
         }
         catch (Exception e)
         {
             log.error(e);
         }

         return null;
     }

    /**
     * is this method cacheable
     */
     public boolean isCacheable()
     {
         return true;
```

```
    }
}

/**
 * a jython velocity GET property
 */
class PyGetProperty implements VelPropertyGet
{
    private static final Logger log = Logger.getLogger(JythonUberspect.class.getName());
    private PyString propname;

    public PyGetProperty(String propname)
    {
        this.propname = new PyString(propname);
    }

   /**
    * the name of the jython property/attribute
    */
    public String getMethodName()
    {
        return propname.toString();
    }

   /**
    * returns the property value
    */
    public Object invoke(java.lang.Object o)
    {
        PyObject po = (PyObject)o;
        try
        {

            Object rtn = po.__findattr__(propname);
            if (rtn instanceof PyNone)
            {
                // handle python None correctly
                return null;
            }
            else
            {
                return rtn;
            }
        }
        catch (Exception e)
        {
            log.error(e);
        }

        return null;
    }

   /**
    * is this property cacheable
    */
    public boolean isCacheable()
    {
        return true;
    }
}

/**
 * a jython velocity SET property
 */
class PySetProperty implements VelPropertySet
{
    private static final Logger log = Logger.getLogger(JythonUberspect.class.getName());
    private PyString propname;

    public PySetProperty(String propname)
    {
```

```
        this.propname = new PyString(propname);
    }

   /**
    * the name of the property/attribute
    */
    public String getMethodName()
    {
        return propname.toString();
    }

   /**
    * set the value of a property
    */
    public Object invoke(Object o, Object arg)
    {
        PyObject po = (PyObject)o;
        try
        {
            if (arg instanceof String)
            {
                po.__setattr__(propname, new PyString((String)arg));
            }
            else if (arg instanceof PyObject)
            {
                po.__setattr__(propname, (PyObject)arg);
            }
            else
            {
                log.error("unsupported argument type : " + arg.getClass().getName());
            }
        }
        catch (Exception e)
        {
            log.error(e);
        }

        return null;
    }

   /**
    * is this property cacheable
    */
    public boolean isCacheable()
    {
        return true;
    }
}

/**
 *  <p>
 *  An Iterator wrapper for a PySequence.
 *  </p>
 *  <p>
 *  WARNING : this class's operations are NOT synchronized.
 *  It is meant to be used in a single thread, newly created
 *  for each use in the #foreach() directive.
 *  If this is used or shared, synchronize in the
 *  next() method.
 *  </p>
 *
 * @author <a href="mailto:jvanzyl@apache.org">Jason van Zyl</a>
 * @author <a href="mailto:geirm@apache.org">Geir Magnusson Jr.</a>
 * @author Kent Johnson
 */
class PySequenceIterator implements Iterator
{
    /**
     * The objects to iterate.
     */
    private PySequence seq;
```

```
    /**
     * The current position and size in the array.
     */
    private int pos;
    private int size;

    /**
     * Creates a new iterator instance for the specified array.
     *
     * @param array The array for which an iterator is desired.
     */
    public PySequenceIterator(PySequence seq)
    {
        this.seq = seq;
        pos = 0;
        size = seq.__len__();
    }

    /**
     * Move to next element in the array.
     *
     * @return The next object in the array.
     */
    public Object next()
    {
        if (pos < size )
            return seq.__getitem__(pos++);

        throw new NoSuchElementException("No more elements: " + pos +
                                         " / " + size);
    }

    /**
     * Check to see if there is another element in the array.
     *
     * @return Whether there is another element.
     */
    public boolean hasNext()
    {
        return (pos < size );
    }

    /**
     * No op--merely added to satify the <code>Iterator</code> interface.
     */
    public void remove()
    {
        throw new UnsupportedOperationException();
    }
}
```

The following Jython class can be used to get an element by index from a list. You would put the listtool into the context (in your jython servlet) and then use it in the template as such: $listtool.get($myjythonlist, 4)

```
class listtool:
    def get(self, obj, idx):
        if type(obj) in (types.TupleType, types.ListType) and idx < len(obj):
            return obj[idx]
        else:
            return None
```