StrutsCatalogLazyList

There are two frequent issues the confront people when dealing with populating a collection of beans in an ActionForm.

- How can I generate the appropriate html using the struts taglibs?
- I have a request scoped ActionForm and am getting an "index out of range" error when I submit the form - what do I do?

More info on indexed properties available in the FAQs

The *Indexed Properies* section below deals with generating the html and the *Lazy List* section shows possible solutions for avoiding the "index out of range" error.

Indexed Properties

Struts html tags have an *indexed* attribute which will generate the appropriate html to populate a collection of beans when the form is submitted. The *trick* is to name the **id** attribute to the same as the indexed property.

For example the following jsp...

```
<logic:iterate name="skillsForm" property="skills" id="skills">

    <html:text name="skills" property="skillId" indexed="true"/>

</logic:iterate>
```

...will generate the following html

```
 <input type="text" name="skills[0].skillId value="..."/>
 <input type="text" name="skills[1].skillId value="..."/>
 ....
 <input type="text" name="skills[n].skillId value="..."/>
```

When the form is submitted BeanUtils will first call the getSkills(index) method to retrieve the indexed bean followed by setSkillId(..) on the retrieved bean.

Using nested tags

An easy-to-use alternative to the html iterate tag and indexed attribute are the nested tags. The nested tag equivalent of the example above would be:

```
<nested:iterate property="skills">
    <nested:text property="skillId"/>
</nested:iterate>
```

The generated html would be the same.

N.B. More info on using ***nested*** tags is available in theJavaDocs and tutorials available at www.keyboardmonkey.com.

Lazy List Behaviour

A common problem with indexed properties, is that people then get "index out of range" errors with ActionForms that are in Request scope. The indexed property (List or Array) needs to be able to automatically *grow* during the ActionForm population process. The key to achieving this is in the getXXXX (index) method.

The following sections show three examples of how to achieve lazy list processing.

BeanUtils Indexed Properties Issue

There is an issue in Commons BeanUtils with indexed properties if you use java.util.List rather than Arrays. Full details about the issue are in Bug 28358.

If you define the following types of getter/setter methods for your indexed property, in JDK 1.3 you shouldn't have any problem, but in JDK 1.4 the getter /setter which use the java.util.List are ignored by Bean Utils.

```
public SkillBean getSkills(int index)
public List getSkills()
public void setSkills(int index, SkillBean skill)
public void setSkills(List skills)
```

The solution is either to use DynaBeans which don't have any issues or use method signatures with Arrays rather than java.util.List. For example the following should work...

```
public SkillBean getSkills(int index)
public SkillBean[] getSkills()
public void setSkills(int index, SkillBean skill)
public void setSkills(SkillBean[] skills)
```

Hand Cranking lazy List in the ActionForm

```
public class SkillActionForm extends ActionForm {

    protected List skills = new ArrayList();

    public List getSkills() {
        return skills;
    }

    // non-bean version so as not to confuse struts.
    public void populateSkills(List skills) {
        this.skills.addAll(skills);
     }

    public void setSkills(SkillBean skill) {
        this.skills.add(skill);
    }

    public SkillBean getSkills(int index) {

        // automatically grow List size
        while (index >= skills.size()) {
            skills.add(new SkillBean());
        }

        return (SkillBean)skills.get(index);

    }

}
```

Using Commons Collections for lazy Lists

Commons collections have a lazy list decorator which automatically grows the list when the get(int) method is called. In the example below the ActionForm implements the Factory interface, providing the create() method to populate an entry in the List.

```
import org.apache.commons.collections.Factory;
import org.apache.commons.collections.list.LazyList;

public class SkillActionForm extends ActionForm, implements Factory {

    protected List skills = LazyList.decorate(new ArrayList(), this);

    public List getSkills() {
        return skills;
    }

    public void setSkills(List skills) {
        this.skills = skills;
    }

    public SkillBean getSkills(int index) {
        return (SkillBean)skills.get(index);
    }

    // 'Factory' method for LazyList
    public Object create() {
        return new SkillBean();
    }

}
```

The syntax in Commons-Collections 2.1.1 is;

```
import org.apache.commons.collections.Factory;
import org.apache.commons.collections.ListUtils;

public class SkillActionForm extends ActionForm, implements Factory {

    protected List skills = ListUtils.lazyList(new ArrayList(), this);

    public List getSkills() {
        return skills;
    }

    public void setSkills(final List newSkills) {
        skills = ListUtils.lazyList(newSkills, this);
    }

    // 'Factory' method for LazyList
    public Object create() {
        return new SkillBean();
    }

}
```

An easier/cleaner way? (Rick Reumann)

Someone feel free to edit this, but doesn't that solution above tie your ActionForm to just one List? What if your form has several lists that need to grow?

Someone gave me this solution and it works great. Since reset is always called it makes sure you have your LazyList initialized and then it just grows as needed. You can set this up for other Lists you need in your form as well.

```
import org.apache.commons.collections.Factory;
import org.apache.commons.collections.ListUtils;
import java.util.List;
import java.util.ArrayList;

public class SkillActionForm extends ActionForm  {

    protected List skills;

    public List getSkills() {
        return skills;
    }

    public void setSkills(final List skills) {
        this.skills =  skills;
    }

    public void reset(ActionMapping actionMapping, HttpServletRequest httpServletRequest) {
      skills = ListUtils.lazyList(new ArrayList(),
      new Factory() {
          public Object create() {
              return new Skills();
          }
      });
  }

}
```

LazyDynaBean / LazyValidatorForm

N.B. Solutions here require **Struts 1.2.4** and **Bean Utils 1.7.0**

LazyDynaBean in Commons BeanUtils has the *lazy* List type processing. LazyValidatorForm is built using the BeanUtils LazyDynaBean and can be used by simply configuring it through the struts-config.xml.

```
<form-beans>

   <form-bean name="skillForm" type="org.apache.struts.validator.LazyValidatorForm>
      <form-property name="skills" type="java.util.ArrayList"/>
   </form-bean>

</form-beans>
```

In fact using Arrays (rather than Lists) a LazyDynaBean can be used directly, in the following way:

```
<form-beans>

   <form-bean name="skillForm" type="org.apache.commons.beanutils.LazyDynaBean">
      <form-property name="skills" type="myPackage.SkillBean[]"/>
   </form-bean>

</form-beans>
```

Struts 1.2.4 will *wrap* the LazyDynaBean in a BeanValidatorForm automatically.

Using the Decorator Pattern Underlying LazyList with Instrumented Forms : One Alternative Suggested Some Time Ago

Rather than use collections classes that are decorated in forms, why not make the forms themselves instrumentable by having them decorate the collections? I suggested this sometime ago on this wiki, but the idea could not be implemented until the processing for forms with the collections classes caught up with the idea, distinguishing between a mere Map, for example, and an ActionForm that was also a decorated Map. I called these Instrumented Forms and used the recommendations of Joshua Bloch to code them. ["http://www.michaelmcgrady.com" – see the coding ideas section and then instrumented forms]

Michael McGrady

LazyDynaBean Gotchas

The above solution using LazyDynaBean looks like a simple drop in replacement which is great, however it assumes that the Struts Validator is active. If you are not familiar with the validator (I wasn't when I got directed to this page for a solution) then it can cost you a number of hours of head scratching and frustration whilst you figure it out. To summarise and hopefully give someone a kick start to this, you need to:

1. Include the Commons Validator jar in your project.

2. Modify your struts-config.xml to include the validator plugin.

3. Copy the the validator-rules.xml file from the struts download or source one from somewhere else.

4. Create and amend the valilation.xml file to include a section for each of your forms you wish to use LazyDynaBeans for.

I found the struts doco assumed you already know how all of this worked so wasn't much help to getting started. The best place to find help is in referenced resources down the bottom of http://struts.apache.org/userGuide/dev_validator.html

Derek Clarkson

Extending the struts DynaActionForm

I came up with this code after looking around and not really being satified with any of the solutions I found. They either required a lot of fiddling and setup in the application, or involved install all sorts of packages I didn't really want to use. Basically I don't like making things any more complex than they should be. So I came up with this class which extends the struts DynaActionForm to effective lazy load lists and arrays without requiring any additional software or setup over the standard DynaActionForm.

```
package com.dhc.form;

import org.apache.struts.action.DynaActionForm;

import java.lang.reflect.Array;
import java.util.List;

/**
 * This class extends
 * {@link org.apache.struts.action.DynaActionForm DynaActionForm} so that we can
 * fix the issues where array references in the properties are not created and
 * cause index exceptions to be thrown. The intent is to re-code the set method
 * so that when an index value is being set, the arrays are extended as
 * necessary by an automatic process. This should remove the necessity for using
 * one of a number of messy work-arounds being used by the struts community.
 * <p>
 * <b>NOTE</b> after submitting this for comment on java ranch I found out about a
 * similar thing using the Bean utils lazy forms. However when I tried to implement
 * it, I ran into all sorts of problems with implementing. Most notibly they are
 * heavily linked to the use of the Validator which menas you must register all forms
 * with it before you can use the lazy forms. It took me some time to work this out
 * because the error messages that are generated are very vague about what the issue is
 * which makes it difficult to sort out.
 * <p>
 * I also looked at using the commons collection LazyList to drive it. but the issue
 * with that is that it requires the class to provide a factory for generating
 * objects to be stored. This is ok when you know what you are storing but the idea
 * behind a dynaactionform is that you do not.
 * <p>
 * To cut a long story short this code is smaller, requires no implementation, xml files
 * supporting it or other libraries.
 *
 *
 * @author Derek Clarkson
 */
public class LazyDynaActionForm extends DynaActionForm {

   // @Override
```

```
   public void set(String name, int index, Object value) {

      // get the array container from the hashmap of properties. This should
      // return an array object or a list.
      Object prop = this.dynaValues.get(name);
      Class arrayClass = prop.getClass();

      // error trap.
      if (prop == null) {
         throw new NullPointerException("No indexed value for '" + name + "[" + index + "]'");
      }

      if (arrayClass.isArray()) {

         // First check that the array is big enough. If not then we need to
         // expand it and store the expanded one.
         if (Array.getLength(prop) < index + 1) {
            Object newArray = Array.newInstance(arrayClass.getComponentType(), index + 1);
            System.arraycopy(prop, 0, newArray, 0, Array.getLength(prop));
            this.dynaValues.put(name, newArray);
            prop = newArray;
         }

         // Now store the value.
         Array.set(prop, index, value);

      } else if (prop instanceof List) {

         // Quick local ref for ease of coding.
         List list = (List) prop;

         // Now check the length and expand as necessary.
         if (list.size() < index + 1) {

            // I could not see any way of doing this other than this. Basically
            // because my criteria was
            // that I did not want to replace the list, only expand it because
            // then I would not have to deal
            // with issues of getting the correct type, etc. Alternatives welcome.
            for (int i = list.size(); i <= index; i++) {
               list.add(null);
            }
         }

         // Store the new value.
         list.set(index, value);

      } else {
         throw new IllegalArgumentException("Non-indexed property for '" + name + "[" + index + "]'");
      }

   }
}
```

Enjoy, Derek Clarkson

Problem - How do we deal with Lists of objects that contain Lists?

I'm still stumped with the best way to handle cases where your nesting goes a bit deeper than the examples shown here. For example lets assume you want to edit on a single form a list of Employees - where you want to be able to edit their name AND a List of phone numbers:

```
Employee              Phone Numbers
--------              -------------
John Doe              888-888-8888
                      888-111-1111
                      222-222-2222
Bill Boo              111-111-3333
                      444-333-3333
```

So we might have basically a List of Employee objects where Employee has:

```
Employee:
   String name;
   List phoneNumbers;

Our form bean can have:

ActionForm:
   List employees;
```

Here's the problem.... in the reset method you can set a LazyList to wrap the Employees but there is still the problem of the internal List of phoneNumbers in each Employee.

I was trying:

```
public void reset(ActionMapping actionMapping, HttpServletRequest httpServletRequest) {

    employees = ListUtils.lazyList(new java.util.ArrayList(), new Factory() {
        public Object create() {
            return buildEmployee();
        }
    });
 }

private Employee buildEmployee() {
    //This really isn't working, I'll still get index out of bounds, when
    //trying to set an index of the phoneNumbers list
    Employee emp = new Employee();
    List phoneNumbers = ListUtils.lazyList(new java.util.ArrayList(), new Factory() {
        public Object create() {
            return new String();
        }
    });
    emp.setPhoneNumbers( phoneNumbers );
    return emp;

}
```

Of course all of these problems go away if I just use the Session to store the form, but I really want to figure out a nice way to get this to work using the Request. Feel free to delete this once a nice solution is proposed.

Somewhat of a solution:

The problem above is that I was using a List of Strings (phoneNumbers) as the property in my Employee object. Everything ends up working fine if instead I create a List of "Contact" objects (where each Contact object has a "phoneNumber" - and I also added a "label"):

```
Employee:
   String name;
   List contacts;

Contact:
   String label;
   String phoneNumber;

Form bean still with:

ActionForm:
   List employees;
```

The reset was then modified to:

```
public void reset(ActionMapping actionMapping, HttpServletRequest httpServletRequest) {

    employees = ListUtils.lazyList(new java.util.ArrayList(), new Factory() {
        public Object create() {
            return buildEmployee();
        }
    });
 }

private Employee buildEmployee() {
    Employee emp = new Employee();
    List contacts = ListUtils.lazyList(new java.util.ArrayList(), new Factory() {
        public Object create() {
            return new Contact();
        }
    });
    emp.setContacts( contacts );
    return emp;
}
```

Still not very satisified

I'm still not happy with the solutions provided (including my own). Many of the above solutions assume you know how deep your items are nested, or that you create nested objects up front that can handle the lazy situation. Where I'm not happy is where you can get back a nested structure of unknown depth. I have a case now where the generated content that can be editable is a questionnaire and each question can end up having sub-questions. Other than using Session scope I'm not sure of the best approach to use. I'm messing around with LazyDynaMaps to see if that might work. Not the ideal solution since I'd prefer to use straight value objects from the midtier, but I'll compromise if need be.

NestedLazyBean does n level indexed properties. For straight forward object graphs (e.g. `person.address.line1`) where you want an "address" DynaBean automatically created when `person.get("address").set("line1", ...)` is called then its slightly more problematic, since how do you know when it hits `get("address")` that it should be a DynaBean property rather than some other regular kind? Using BeanUtils's ***Map*** *notation (i.e. using `person[address][line1]`) works from a LazyDynaMap PoV, but you will need the BeanUtils 1.8.0-BETA, since nested maps were not supported until that version.*