EL vs Dependency Injection

By Roger Keays, 17 July 2007

One of the many useful utilities in Java EE is dependency injection - particularly for obtaining an EntityManager for your persistent classes. However, as I've alluded in previous posts, my preference is still for the plain old Servlet stack with JPA, JSF and EL dropped in as jar files. That means no dependency injection for me!

I am aware that there are existing inversion of control containers which might be able to solve this particular problem and which are certainly buzz-word compliant. Unfortunately, I couldn't really see that all the additional configuration was actually simplifying the job for me.

So, is there a another way?...

My alternative solution has been to go back to the traditional reference-by-name paradigm we all used with JNDI, except by replacing JNDI with EL. Using EL from Java allows me not only to 'inject' EntityManagers, but also any configuration variable or bean which can be accessed from the ELContext. When coupled with the pluggable EL resolver, its actually hard to think of use cases where I *couldn't* use EL.

Here is some sample code to resolve an EntityManager:

```
public List<Object> getObjects() {
    EntityManager em = eval("${em}", EntityManager.class);
    return em.createQuery("SELECT e FROM Entity e").getResultList();
}
```

In my code, I'm simply using a RequestFilter to insert the ${em} into the request scope, although you could equally use a custom ELResolver to resolve this variable. The eval() method is a static import which looks like this:

```
/**
 * Shorthand for Application.evaluateExpressionGet(...) which
 * automatically casts the result to expected type and uses the current
 * FacesContext for evaluating the expression.
 *
 * @param expression the EL expression to evaluate
 * @param clazz the expected resultant class
 */
 public static <T extends Object> T eval(String expression,
```

```
        Class<T> clazz) {
    return (T) FacesContext.getCurrentInstance().getApplication().
            evaluateExpressionGet(FacesContext.getCurrentInstance(),
            expression, clazz);
 }
```

There are also fewer limitations on where EL can be used compared to dependency injection. It can be used whenever an ELContext is available, and in a JSF application that is practically anywhere. I use EL for configuring beans which avoids the use of static variables that can break an application when a shared classloader is being used.

```
/** remove the file from the filesystem when deleting */
@PreRemove public void deleteFile() {
    String dir = eval("${config['uploadsDir']}", String.class);
    File file = new File(dir, id.toString());
    file.delete();
}
```

Using EL also makes your code more readable, and your beans scope-independent. Here is a snippet of the first few lines of an ActionListener from some code which doesn't use EL:

```
/* initial values */
context = FacesContext.getCurrentInstance().getExternalContext();
item = (Content) context.getRequestMap().get("item");
site = (Site) context.getApplicationMap().get("site");
em = (EntityManager) context.getRequestMap().get("em");
```

and the equivalent using EL:

```
/* initial values */
item = eval("${item}", Content.class);
site = eval("${site}", Site.class);
em = eval("${em}", EntityManager.class);
```

Probably my favourite advantage of using EL is that it simplifies the problem of referencing items in an iterated context. For example, given the following template:

```
<h:dataTable value="${items}" var="${item}">
  <h:column>
    <h:commandButton action="delete" value="Delete this Item"/>
  </h:column>
```

```
  ...
</h:dataTable>
```

You can resolve the selected item in your action simply by doing this:

```
Object item = eval("${item}");
```

There doesn't seem to be much opinion out on using EL from within Java. I've found it a really great way to make my code more readable and patterns more repeatable. Personally, I'd love to see it become a part of the language!

What do you think? I've given some of the advantages of EL, how about some disadvantages? Performance anybody? Type safety? Tell me why I shouldn't keep using this pattern.

Note: the eval() method shown above is available in the Furnace Webapp Framework.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.