How do we repopulate controls when validation fails

When validation fails, we typically forward back to the same server page, so that the errors can be presented, and so that the client can fix the problem and resubmit the form. Of course, aside from the errors, we may also need to present rich controls, like drop down lists.

If we try to populate rich controls in an `Action` method, like `input` or `execute`, and validation fails, the method will not be invoked, and the controls are not populated. Two alternative ways to populate controls are the `Preparable` interface and the `action` tag.

Preparable interface

Instead of populating controls in an `Action` method, implement the `Preparable` interface, and use a `prepare` method instead. The `prepare` method is called before validation, so it validation fails, we still have a chance to populate controls (or whatever).

Input.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<s:form>
<s:select
   tooltip="Choose Your Favorite Language"
   label="Favorite Language"
   list="languages"
   name="language"
   listKey="key"
   listValue="description"
   emptyOption="true"
   headerKey="None"
   headerValue="None"/>
<s:submit>
</form>
```

Input.java (prepare)

```
    public String prepare() {
        languages.add(new Language("EnglishKey", "English Language"));
        languages.add(new Language("FrenchKey", "French Language"));
        languages.add(new Language("SpanishKey", "Spanish Language"));
        return SUCCESS;
    }

    List languages = new ArrayList();
    public List getLanguages() {
        return languages;
    }

    String language;
    public void setLanguage(String value) {
        language = value;
    }
    public String getLanguage() {
        return language;
    }

    public static class Language {

        public Language(String key, String description) {
            this.key = key;
            this.description = description;
        }

        String key;
        public String getKey() {
            return key;
        }

        String description;
        public String getDescription() {
            return description;
        }
    }
}
```

If a custom stack is being used, be sure to put the Prepare Interceptor before the Validation Interceptor.

action tag

Another solution is to use the action tag to execute an Action in place.

One way to use this tag is to put a control on a "snippet" JSP that is rendered as a result of an Action that does nothing but create the object that populates the control. The action tag sets "executeresult=true", then control markup will be "included" into the page (like a tile), after the action executes.

In effect, `exectuteResult` actions can be used like a tag that can run its own action before emitting the markup.

Input.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<s:form>
   <s:action name="Languages" executeResult="true"/>
   <s:submit/>
</s::form>
```

Languages.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<s:select
   tooltip="Choose Your Favorite Language"
   label="Favorite Language"
   list="languages"
   name=language"
   listKey="key"
   listValue="description"
   emptyOption="true"
   headerKey="None"
   headerValue="None"/>
```

Languages.java

```
public class Languages extends ActionSupport {
    public String execute() {
        languages.add(new Language("EnglishKey", "English Language"));
        languages.add(new Language("FrenchKey", "French Language"));
        languages.add(new Language("SpanishKey", "Spanish Language"));
        return SUCCESS;
    }

    List languages = new ArrayList();
    public List getLanguages() {
        return languages;
    }

    public static class Language {
        String description;
        String key;

        public Language(String key, String description) {
            this.key = key;
            this.description = description;
        }

        public String getKey() {
            return key;
        }

        public String getDescription() {
            return description;
        }

    }


}
```

struts.xml (Input, Languages)

```
<action name="Input">
    <result>/app/Input.jsp</result>
</action>
<action name="Languages" class="app.Languages">
    <result>Languages.jsp</result>
</action>
```

The advantage being that the "Languages" action could be dropped in wherever the "Languages" control is needed, and that the Action for the form doesn't need to know how to populate the Languages control.

Now, the Action is going to be hit every time the page is rendered, but so long as you are using a caching data access layer, like IBATIS or Hibernate, it will end up being a memory-to-memory transfer, rather than a database access.