UI Adaptor Service

Summary

We'll examine interlocking between the e-government standard framework and UI solution(Rich Internet Application).
The type of applying UI Adaptor is difficult to be standardized with one specific method.
In general, the most common method of interlocking the Web Framework and UI solution is the method of changing the data to DTO type before calling the business logic in the Servlet object that performs the Controller role, and transferring to the business logic.

The e-government standard framework is mapping the method in the Controller class and URI requested at the time of development based on Spring MVC Annotation.
Accordingly, it selects the **type of guiding the object coming to the parameter of method to the business DTO class rather than request object.**
(In fact, it is same as using @ModelAttribute.)
However, it is required to define and apply the suitable structure in consideration of the feature of non-functional requirements per project.

Detailed Guide per UI Solution Company

- MyPlatform_Interlock_Sample_ManualIncluded_.zip - written by KoSeokRyul(MiPlatform)
- Tomato Systems – Tomato Systems
- SHIFT - SHIFT(Gauce)

Description

What we will examine intensively is the process of changing the data from the UI solution at the front end of Controller into the DTO.
For data change, we'll examine the **method of using ArgumentResolver**.

Data objects are transferred to UI solution with inclusion in request object. What we need is a business DTO class.
The Business DTO class exists as the parameter of mapped Controller method and URI(@RequestMapping).
AnnotationMethodHandlerAdaptercalls ArgumentResolvers(including customArgumentResolvers) corresponding to the class set in the parameter of Controller method (DTO, here).

Accordingly, we expand ArgumentResolver to develop CustomRiaArgumentResolverand register in AnnotationMethodHandlerAdapter.
Object returned at CustomRiaArgumentResolveris used as a parameter of method at Contorllercolumn.

* Note :AnnotationMethodHandlerAdapterexecutes and imports ArgumentResolver for the object type that exists as the parameter when executing the method of controller mapped and URI.

And, the execution results of Controller column is transferred to RiaView through ViewResovler and RiaVeiw converts the outcome, DTO, to UI solution data type and sends to the response.

In summary, following contents can be summarized as follows:

1. Register CustomRiaArgumentResolver of AnnotationMethodHandlerAdapter⇒**CustomRiaArgumentResolver**
2. Register UIAdaptor⇒**UIAdaptorImpl**
3. Register RiaView⇒**RiaView**

We'll regard and examine the DTO used as the following class.

UdDTO.java(sample)

```
...
public class UdDTO implements Serializable {

privateMapvariableList ;
privateMapdataSetList ;
privateMap Objects ;

public void setVariableList(MapvariableList) {
          this.variableList= variableList;
     }

public void setDataSetList(MapdataSetList) {
          this.dataSetList= dataSetList;
     }

publicMapgetVariableList() {
returnvariableList;
     }

publicMapgetDataSetList() {
returndataSetList;
     }

public void setObjects(Map objects) {
          Objects = objects;
     }

publicMapgetObjects() {
          return Objects;
     }
}
...
```

Convert to DTO at UI Solution Data

Next, we'll explain the process of getting UTO and converting it to UTO from the object by UI solution. UIAdaptorImplobject, in charge of conversion, is set in **CustomRiaArgumentResolver** ofAnnotationMethodHandlerAdapter.

Setting Information (CustomRiaArgumentResolver)

```
          <bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
                    <property name="webBindingInitializer">
                              <bean class="egovframework.rte.fdl.web.common.EgovBindingInitializer"
/>
                    </property>
                    <property name="customArgumentResolvers">
                              <list>
                                        <bean
class="egovframework.rte.fdl.sale.web.CustomRiaArgumentResolver">
                                                  <property name="uiAdaptor">
                                                            <ref bean="riaAdaptor" />
                                                  </property>
                                        </bean>
                              </list>
                    </property>
          </bean>
```

CustomRiaArgumentResolver, the implement of WebArgumentResolver, executes Adapter set in uiAdaptor.

CustomRiaArgumentResolver.java

```
public class CustomRiaArgumentResolver implements WebArgumentResolver {

    privateUiAdaptoruiA;

    public void setUiAdaptor(UiAdaptoruiA) {
        this.uiA = uiA;
    }

    publicObjectresolveArgument(MethodParametermethodParameter,
NativeWebRequestwebRequest) throws Exception {

        Class<?> type = methodParameter.getParameterType();
        ObjectuiObject = null;

        if (uiA == null)
            return UNRESOLVED;

        //Parameter type information of the method executed of Controller is transferred
through MethodParameter.
        //Compare with UTO registered in UIAdaptroimplement set in advance.
        if (type.equals(uiA.getModelName())) {
            HttpServletRequest request = (HttpServletRequest)
webRequest.getNativeRequest();
            // here, create and transfer data.
            uiObject = (UdDTO) uiA.convert(request);
            returnuiObject;
        }

        return UNRESOLVED;
    }
...
```

Implement of UiAdaptoris as follows. Here, for example, the code of Miplatformwas created.
The role of moving the data from the object of UI solution to DTO was performed at converte4In method.

RiaAdaptorImpl.java(MiPlatform⇒UdDTO)

```
public class RiaAdaptorImpl implements UiAdaptor {

protected Log log = LogFactory.getLog(this.getClass());


  //Method called at resolveArgumentmethod
    publicObject convert(HttpServletRequest request) throws Exception {

        PlatformRequestplatformRequest = null;

        try {
            platformRequest = new PlatformRequest(request,
PlatformRequest.CHARSET_UTF8);
            platformRequest.receiveData();
        } catch (IOException ex) {
            ex.getStackTrace();
            // throw new IOException("PlatformRequest error");
        }
   //Conversion from UI solution data to DTO object
        UdDTOdto = converte4In(platformRequest);
        returndto;
    }
```

```
          privateUdDTO converte4In(PlatformRequestplatformRequest) {
                    UdDTOdto = new UdDTO();
                    //... Fill in DTO or VO value
                    .....
                    returndto;
          }

          public Class getModelName() {
                    returnUdDTO.class;
          }
}
```

Implement Controller Method

UdDTO class is created at CustomRiaArgumentResolver and is brought in the form of parameter of Controller method.
The following example is the method at Controller end.

XXCategoryController.java

```
...
          @RequestMapping("/sample/miplatform.do")
          publicModelAndView selectCategoryList4Mi(UdDTOmiDto, Model model) throws Exception {

                    ModelAndViewmav = new ModelAndView("riaView");

                    //Map to be used if there is a search condition
                    Map<String, String>smp = new HashMap<String, String>();
                    try {

                      //Call Biz Layer.
                              ListresultList = categoryService.selectCategoryList(smp);

                              //Save the result value in the model
                              mav.addObject("MiDTO", resultList);

                    } catch (Exception ex) {
                              log.info(ex.getStackTrace(), ex);
                    }
                    returnmav;
          }
```

BeanNameViewResolverSetting

The name of model object is riaView. It is direct indication of Bean Name and requires setting as shown below(BeanNameViewResolver).

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver" p:order="0" />

<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver"
                    p:order="1" p:viewClass="org.springframework.web.servlet.view.JstlView"
                    p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

<bean id="riaView" class="egovframework.rte.fdl.sale.web.RiaView" />
```

Implement RiaView

Code of RiaView is as follows. It is the module to convert to export DTO again to match to the company.
Here, it is converted to Miplatform object. It is the logic to send in the form of stream after

instantiating it to DataSet called egovDs.
Accordingly, change it to convert and send it in the form of data per company.

RiaView.java(DTO ⇒MiPlatform)

```
....
public class RiaView extends AbstractView {

		protected Log log = LogFactory.getLog(this.getClass());

		@SuppressWarnings("unchecked")
		@Override
		protected void renderMergedOutputModel(Map model, HttpServletRequest request,
HttpServletResponse response)
		throwsException {
				VariableListmiVariableList = new VariableList();
				DatasetListmiDatasetList = new DatasetList();
				PlatformDataplatformData = new PlatformData(miVariableList, miDatasetList);

				Listlist = (List) model.get("MiDTO");

				Iterator<Map> iterator = list.iterator();
				Iterator<Map>dataIterator = list.iterator();

				Dataset dataset = new Dataset("egovDs");

				while (iterator.hasNext()) {
						// Header setting
						Map<String, Object> record = iterator.next();
						Iterator<String>si = record.keySet().iterator();

						while (si.hasNext()) {
								String key = si.next();
								dataset.addColumn(key, ColumnInfo.COLUMN_TYPE_STRING,
(short) 255);
						}
				}

				while (dataIterator.hasNext()) {
						Map<String, Object> record = dataIterator.next();
						Iterator<String>si = record.keySet().iterator();
						// Header setting
						while (si.hasNext()) {
								String key = si.next();
								dataset.addColumn(key, ColumnInfo.COLUMN_TYPE_STRING,
(short) 255);
						}

						// Value setting
						int row = dataset.appendRow();
						Iterator<String> si2 = record.keySet().iterator();
						while (si2.hasNext()) {
								String key = si2.next();
								String value = (String) record.get(key);

								System.out.println("key = " + key + " , value = " + value);
								dataset.setColumn(row, key, value);
						}
						miDatasetList.add(dataset);
				}
				try {
```

```
                              newPlatformResponse(response,
PlatformConstants.CHARSET_UTF8).sendData(platformData);

                    } catch (IOException ex) {
                              if (log.isErrorEnabled()) {
                                        log.error("Exception occurred while writing xml to MiPlatform
Stream.", ex);
                              }

                              throw new Exception();
                    }

          }

}
```

Reference