As jspx framework aims to reduce and eliminate the complexity in web development, one of the big advantages it introduces is the mechanism of transferring data between the controller layer and the view layer. This is done using JspxBean; JspxBean is an annotation used to tag your java bean as a data-transfer-object between your controller layer and your view layer. JspxBean is considered the most efficient and clean way of transferring data between different controllers as well.

Defining & Using JspxBean

Unlike other web frameworks (and as jspx main focus of reducing complexity and configurations), no need to define your JspxBean in any configuration file. To tag your java bean as data-transfer object, you need to add the JspxBean annotation above the definition of the attribute in the page controller. Public getters and setters methods need to be defined in the controller for each JspxBean.

```
@JspxBean(name="customer",scope=JspxBean.SESSION)
private Customer customer;
```

The JspxBean annotation expect three attributes defined below

Attribute	Type	Description
name	String	The name of the JspxBean to be used in the view
Scope	int	The scope of the JspxBean: request, session or conversation
dateformat	string	The data format used to render date objects (Optional as the default is set to dd-MM-yyyy)

As mentioned above, there is no configuration for JspxBeans; JspxBean can be bound to the view as following

```
${beanName.propertyName}
```

Where bean name represents the name supplied to the JspxBean annotation in the controller, and the propertyName is the simple java name of your bean attribute. As the annotated bean is following the standard java beans, public getter methods are used to get the required value. The lines below show how to use the defined JspxBean in the view:

```
<label>${customer.name}</label>
```

In the above example, jspx framework will evaluate the above expression by invoking the `public getName()` method in the Customer object. If the name of the customer is Ali; the following will be the final HTML sent to the browser:

```
<label>Ali</label>
```

Note: Expression language is not supported as this phase, so you can't use JspxBean in formulas (i.e. jspx framework will not be able to evaluate this expression ${customer.id + 5})

Binding JspxBeans

JspxBeans can be bonded to the following:

- Attribute of any control (i.e. <input type="text" value="${customer.id}"/>)
- Content of a label (i.e. <label>${customer.name}</label>)

At the rendering phase, the framework evaluates jspx expressions by invoking the public getters of the bean. The framework takes care of the conversion of the returned value to String; all java primitive types (i.e. int, double, long, etc) are supported in addition to the conversion of Date objects using the dateformat of the annotated JspxBean.

As JspxBeans can be use to transfer data from the controller to the view page (rendering), the JspxBean can also be used to transfer data in the opposite direction (page to controller). This is done by binding the form controls value attribute to the JspxBean, for example:

```
<input type="text" value="${customer.id}"/>
```

When the page form is submitted the framework will automatically parse the input control value from the posted request parameters and set it in the customer JspxBean by invoking the `public setId(int id)` method of the customer object.

Complex Objects Binding

JspxBean don't limit the binding to standard java beans, in fact it supports the binding of complex objects that contains references to other objects.

```
${beanName.property.subProperty1.subProperty2}
```

The above expression will be evaluated by jspx by invoking the getProperty() method in the bean object, then invoking getSubProperty1() method in the returned object of the first call, then invoking getSubProperty2() method in the returned object of the second call. The final value is the value returned by the third call.

For example if the customer object contains an account object with serialNumber of 1234, the following expression

```
<label>${customer.account.serialNumber}</label>
```

Will be evaluated to the following

```
<label>1234</label>
```

Super Class Binding

Unlike other web frameworks, jspx is able to bind super-class attributes as well. For example let's consider IT is a sub-class of Department class. Department class contains an attribute name with public accessor method getName(). According to java inheritance, IT objects expose getName() method without the need to implement it as it is inherited from the super class. The jspx framework will be able to evaluate the following expression:

```
<label>${it.name}</label>
```

When evaluating the above expression, the framework will first try to invoke the getName() in the IT object, if the method doesn't exist, it will try to invoke the getName() method in the super class (Department).

Currently jspx support up to 3 level of inheritance when evaluating super-class binding. That is, it tries to find the method in the bean object, its first super-class, second and third before giving up and consider this method doesn't exist in this object.

Scopes & Transferring Beans between Controllers

JspxBean can be defined in one of two scopes; request or session. The scope defines the lifecycle of the JspxBean. Request-scope JspxBeans are contained within the request lifecycle; when the request is completed, the JspxBean will be destroyed along with other request attributes. In the other hand, session JspxBeans live as long as the session is alive (you can clear the jspx bean from session by calling `clearJspxBean` method in the Page class).

JspxBean can be used to transfer objects/data between different controllers. For example suppose you have two controllers CustomerForm and CustomerView controlling two pages (customerForm and customerView) where a customer object is to be filled in the customerForm and passed to the customerView page to be displayed. JspxBean can be used here by defining a customer bean in both controllers with the same jspx bean name and use this bean in both pages. When customer form submits, the framework will bind the submitted data to the customer bean in the CustomerForm controller. When the CustomerForm controller dispatches to the CustomerView controller, the framework will automatically set the customer bean in the CustomerView controller with the customer bean from the sender (CustomerForm controller). Therefore, at the end the customer bean in the CustomerView controller will be pre-populated with the data submitted by the customerForm view.

The above discussion explained how the request JspxBean can be used to transfer data from one controller to the next controller in the request life cycle. However to transfer data between more than one controller controlling different requests lifecycles, session JspxBean would be the choice. After the page init phase, the framework will get the JspxBean from session and set it in the controller attribute. See JspxBean synchronization for more info.

Synchronizing JspxBean

JspxBeans are stored either in the session or the request as attributes according to the scope defined in the definition of the JspxBean. The objects stored in the session/request are synchronized with the objects in the page controller as following:

1- after pageInit phase and before pagePreLoad phase

JspxBean is read from session/request. If not found, the framework will try to read it from the controller, if it is initialized as null in the controller, the framework will create a new instance of this object. At the end the final object will be set in session/request and the controller.

2- after pageLoading phase and before pageRender:

only in case of post-back requests, and after loading the JspxBean with the data submitted in the request parameters, the framework will set the loaded JspxBean in the controller and in the session/request.

3- after pagePreRender and before pageRender phase

The framework will replace the JspxBean value in the session/request with the one read from the controller.