RESEARCH ARTICLE OPEN ACCESS

Web Based Custom Validation Using Framework in Java

Ankur Saxena
Amity University
Sector – 125, Noida
Uttar Prdesh – India
Asaxena1(at)amity(dot)edu

ABSTRACT
In this type of research, we have discussed custom validation of java Struts2 for security purposes. Custom validation is user defined validation of Struts2. These techniques are light-weight, efficient, and no false positive.it is a part of J2ee simplify the foundation of the enterprise level application program, also because the designers and the programmers to distribute the function in each discreteness of the server end when using J2ee to establish the application programs. Today, traditional desktop applications, such as document viewers, presentation tools and chat applications are commonly available as online JavaScript applications for validations. Previous research on web vulnerabilities has primarily concentrated on flaws in the server-side components of web applications. We implement our technique in Struts2 framework. The main results of this paper are a flexible struts2 framework for custom validation.

Keywords:- Java, J2ee,Web, Validation,Struts2,Interceptor,MVC**.**

I. INTRODUCTION

J2ee (Java 2 Platform, Enterprise Edition) is Sun's preferred Java platform for multi-tier enterprise applications. It simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behaviour automatically, without complex programming [1]. J2ee uses a multi-tier distributed application model. There are three tiers in the typical J2EE application model: Web presentation tier, business logic tier and data tier [2].

Web is the very complex issues these days. Since the desire of the companies and organizations are increasing so the complexity and the performance of the Web programming matters. Complexity with the different types of communication devices is increasing [3]. A typical Web application has two parts: a server-sides component and a client-side component. The server-side component processes the user's request and generates an HTML response that is sent back to the browser. The client-side code of the web application, typically written in JavaScript, is sent with the HTML response from the server. The client-side component executes in the web browser and is responsible for processing input data and dynamically updating the view of web page on the client [4].

The framework is designed to streamline the full development cycle, from building, to deploying, to maintaining applications .it can be considered as a set of functions helping the developers in creating the applications [5].

Standard input validation mechanisms should make sure that all input is validated for length, type, syntax, and business rules before accepting the data to be displayed, stored or used [6]. This task can be repetitive and tedious for a programmer, and this is the primary motive for implementing frameworks for input validation (Commons Validator [7], Struts 2 [8], Hibernate Validator [9] and Heimdall [10]). Such frameworks make it easier to maintain and execute the testing code by decoupling the application logic from the validation logic.

For object-oriented languages like Java, the challenge is to validate specific properties of an object representing the input, without writing validation code in the object itself. Historically, XML configuration files have been used to achieve this separation of concerns, by explicitly storing the names of the properties to be tested and that of the tests to be performed. At runtime, reflection [11] or Servlet filters (listener or interceptors) [8] would then used to actually run the tests on the target methods.

Validation process as it is seen by the user. Only a few lines of code need to be inserted into the application in order to use the framework. Namely, a new Validator object has to be created, and its validation method validate () has to be invoked on an annotated object o to validate it. A Validation Summary object is returned, containing the results of the validation tests for the object [12].

1.1 Struts 2: Struts2 provides supports to POJO based actions, Validation Support, AJAX Support, Integration support to Hibernate and spring

frameworks, support to various result types such as Velocity, JSP etc [13].
A Struts2 provides many features that were not in struts1. The important features of struts2 framework are given below:

1. An Action class implements an Action interface. Struts2 provides a base Action Support class that implements commonly used interfaces and an Action interface is not necessary. Any POJO object along with an execute signature can be used as a Struts2 Action object.
2. Struts2 Actions are not coupled to a container type. Most often the servlet contexts are represented as simple Maps and allowing Actions to be tested in isolation. Struts2 Actions can still access the original request and response, when required. However other architectural elements reduce or eliminate the need to access the HttpServetRequest or HttpServletResponse directly.
3. Struts2 uses Action properties as input properties and eliminating the need for a second input object. All Input properties may be rich object types which may have their own properties. The Action properties can be accessed from the web page via the taglib. Struts2 also supports the Action Form pattern, as well as POJO form objects and POJO Actions. Rich object types, including business objects, can be used as input/output objects. The Model Driven feature simplifies taglib references to POJO input objects
4. Struts2 Actions can be tested by instantiating the Action, setting properties, and invoking methods. Dependency Injection support also makes testing simpler.
5. Struts2 can use JSTL, but it also supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL).
6. Struts2 uses OGNL for type conversion and converters to convert Basic and common object types and primitives as well.
7. Struts2 allows manual validation that is done by using the validate method and the XWork Validation. This Validation Framework allows chaining of validations into sub-properties using the validations defined for the properties class type and the validation context
8. Struts2 uses a Value Stack technology to make the values accessible to the taglibs without coupling the view to the object to which it is rendering. The Value Stack strategy enables us to reuse views across a range of types, having same property name but different property types[14].

1.2 Struts 2 Custom Validation: We can define our own custom validation or validation logic in java struts 2 by implementing the Validateable interface in the action class.
The workflow interceptor is used to get information of the error messages defined in the action class.
The workflow interceptor checks if there is any validation errors or not. It doesn't perform any validation.

It is applied when action class implements the Validateable interface. The input is the default parameter for this interceptor that determines the result to be invoked for the action or field error. It is found in the default Stack so we don't need to define it explicitly.
There is only one parameter defined for workflow interceptor.
inputResultName: this method specifies the result name to be returned if field error or action error is found

Validateabale
interface: The Validateable interface must be implemented to perform validation logic in the action class. It contains only one method validate() that must be overridden in the action class to define the validation logic. Signature of the validate method is:

public void validate();
Validation Aware interface: The Validation Aware interface can accept the action class level error messages. The field level messages are kept in Map and Action class level messages are kept in collection. It should be implemented by the action class to add any error message.

Methods of Validation Aware interface
The methods of Validation Aware interface are as follows:

Method	Description
void addFieldError(String fieldName,String errorMessage)	This Method adds the error message for the specified field.
void addActionError(String errorMessage)	This Method adds an Action-level error message for this action.
void addActionMessage(String message)	This Method adds an Action-level message for this action.
void setFieldErrors(Map<String,List<String>> map)	This Method sets a collection of error messages for fields.
void setActionErrors(Collection<String> errorMessages)	This Method sets a collection of error messages for this action.

void setActionMessages(Collection<String> messages)	This Method sets a collection of messages for this action.
boolean hasErrors()	This Method checks if there are any field or action errors.
boolean hasFieldErrors()	This Method checks if there are any field errors.
boolean hasActionErrors()	This Method checks if there are any Action-level error messages.
boolean hasActionMessages()	This Method checks if there are any Action-level messages.
Map<String,List<String>> getFieldErrors()	This Method returns all the field level error messages.
Collection<String> getActionErrors()	This Method returns all the Action-level error messages.
Collection<String> getActionMessages()	This Method returns all the Action-level messages.

Table 1.1 Methods of Validations.

II. RELATED WORK

In this section, we introduce the running example used throughout the paper, and show how annotations can be used to define tests on single properties of an object. We will use the web form for international money transfers from a hypothetical Internet bank (see Figure 1). IBAN (International Bank Account Number) is the standard for identifying bank accounts internationally (not in USA). Some countries have not adopted this standard, and for money transfer to these countries, a special clearing code is needed in combination with the normal account number of the beneficiary. BIC (Bank Identifier Code), also known as SWIFT. It is needed to identify the beneficiary's bank uniquely.

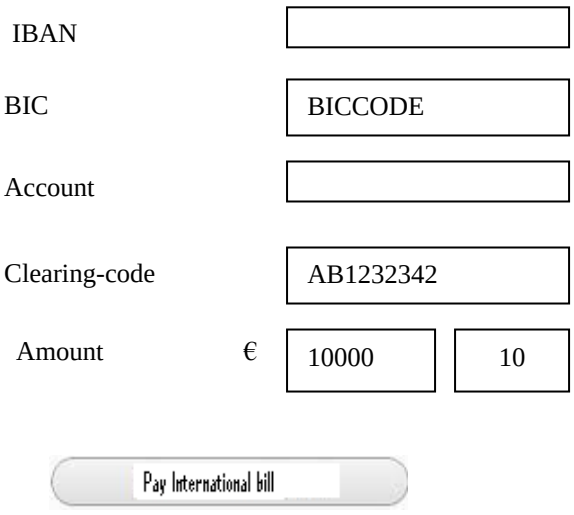

Figure 1

We assume that the object representing the form is created in Java, and that each field in the web form is represented by a property of this object. Fields where the user does not enter a value, are in this example represented by the null value. A partial implementation of this Java object is shown in Figure 2. Here every annotation represents a test to be run on the return value of the method it is applied to. In our framework, annotations representing tests are called *validation-annotations*. This categorization is further split into property-annotations, which represent property-tests, and *cross-annotations*, which represent cross-tests. All the annotations in Figure 2 are property-annotations, i.e., they involve checking a single specific property.

We use property-tests to check whether basic formatting rules are respected. For example, the annotation *@IntRange(min=0,max=10000)* represents a test that checks whether the value of amountEuro is non-negative and not greater than *10000*. The property-annotation *@IntRange(min=0,max=99)* represents a test to check whether amountCents is between *0* and *99*. The propertyannotation *@ValidateBIC* represents a property-test for *BIC* codes, and *@Required* means that the field cannot be left empty.

The annotations only specify what tests should be run on each value. To actually run the tests, an object must be passed to a validator. The validator inspects the object through reflection, extracts the annotations and the return

```
@ValidateBIC
@Required
public String getBIC()
{ return BIC; }

@IntRange(min=0,max=10000)
public Integer getAmountEuro()
{ return amountEuro; }
```

```
@IntRange(min=0,max=99)
public Integer getAmountCents()
{ return amountCents; }
```

Figure 2: Example code using the property-annotations to test input from the web form in Figure 1.

```
@Validation
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.ANNOTATION_TY
PE,
ElementType.METHOD})
public @interface IntRange {
int min();
int max();
public static class Tester
implements IPropertyTester<IntRange,
Integer> {
public boolean runTest(IntRange r,
Integer v) {
return(v >= r.min() && v <= r.max());
}}}
```

Figure 3: Example of property-annotation.

Values from the getter-methods, and invokes the corresponding test [11]

III. IMPLEMENTATION

In this part of paper we will see how we can validate using Struts 2with tomcat (open source servlet container developed by the Apache Software Foundation (ASF))[15]. Lets first create the login page, we use Struts 2 UI tags to create the login page.

The s:form tag contains all the form elements. The action attribute contains the action name to which the form should be submitted. This action name should be same as the one specified in the XML declarative architecture. The struts.xml files do the configuration.

The textfield tag is used to create a text box. The label attribute of the textfield tag contains the name to be displayed on the page and the name attribute contains the name of the property in the action class to be mapped. The password tag is same as the textfield tag except that the input value is masked. The submit tag is used to create a submit button, the value "register"represented the label of the button.

Steps to perform custom validation

The steps are as follows:

1. create the form to get input from the user
2. Define the validation logic in action class by extending the ActionSupport class and overriding the validate method
3. Define result for the error message by the name input in struts.xml file

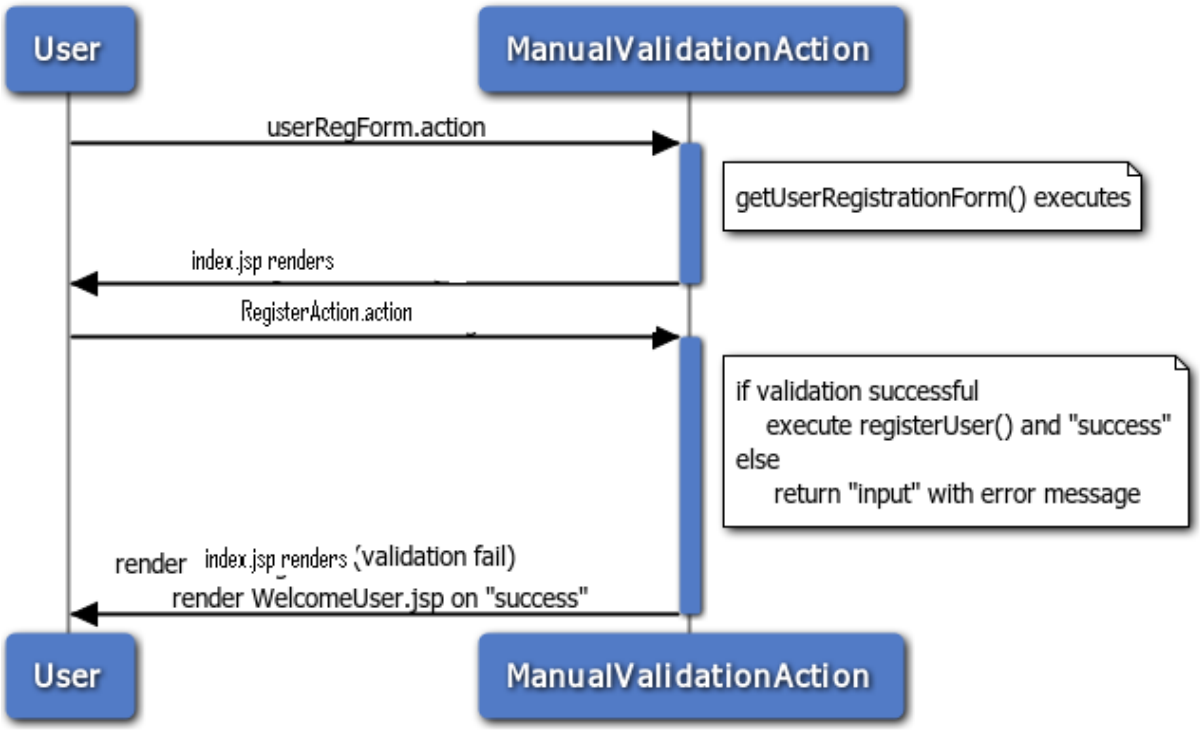

Figure 4 Flow of Custom Validation

For this approach we are creating 4 pages:

1. index.jsp for input from the user.
2. RegisterAction.java for defining the validation logic.
3. struts.xml for defining the result and action.
4. welcome.jsp for the view component.

1) Create index.jsp for input:This jsp page creates a form using struts UI tags. It receives name, password and email id from the user.

index.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:submit value="register"></s:submit>
</s:form>
```

2) Create the action class:This action class inherits the ActionSupport class and overrides the validate method to define the validation logic.

RegisterAction.java

```
package com.ankur;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterAction extends ActionSupport{
private String name,password;
public void validate() {
if(name.length()<1)
addFieldError("name","Name can't be blank");
if(password.length()<6)
addFieldError("password","Password must be greater than 5");
}

public void String setName(String name)
{
this.name=name;
}
 public void String setPassword(String password)
{
this.password=pasword;
}
public String getName()
{
return name;
}
 public String getPassword()
{
return password;
}
 public String execute(){
//perform business logic here
   return "success";
}
```

3) Define a input result in struts.xml:This xml file defines an extra result by the name input, that will be invoked if any error message is found in the action class.

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
 Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
<package name="default" extends="struts-default">
<action name="register" class="com.ankur.RegisterAction">
<result>welcomeUser.jsp</result>
<result name="input">index.jsp</result>
</action>
</package>
</struts>
```

4) Create view component:It is the simple jsp file displaying the information of the user.

welcomeUser.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
Name:<s:property value="name"/><br/>
Password:<s:property value="password"/><br/>
```

Defining action level error message:The action level error message works for the whole form. We can define the action level error message byaddActionError() method of ValidationAware interface in validate() method.

```
package com.ankur;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterAction extends ActionSupport{
private String name,password,email;
public void validate() {
if(name.trim().length()<1 ||
password.trim().length()<1){
addActionError("Fields can't be blank");
}
}
public void String setName(String name)
{
this.name=name;
}
 public void String setPassword(String password)
{
this.password=pasword;
}
```

```
public void String setEmail(String email)
{
this.email=email;
}

public String getName()
{
return name;
}
 public String getPassword()
{
return password;
}
 public String getEmail()
{
return email;
}

public String execute(){
return "success";
}
}
```

Now you need to use action error tag in index.jsp file to display the action level error message.

index.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:actionerror/>
<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:textfield name="email" label="Email Id"></s:textfield>
<s:submit value="register"></s:submit>
 </s:form
```

IV. FUTURE WORK

We have presented a new framework for system design Custom validation .Custom Validation of Struts2 are the best to attract the software developers to work .Custom Validation of Struts 2 are a powerful technology for validate data, and it enables Application to access data from any source in a platform-independent manner. Struts framework is a best implementation of MVC based architecture .Future work of this report is to develop an enterprise application which is based on Struts, Spring and Hibernate Custom Validation and we plan to implement our solution on the .NET framework and also in PHP, as well as incorporate automated injection error logging

V. CONCLUSION

The main idea in the design of this framework has been that it should be easy to create libraries of custom validation, and that these tests should be highly reusable.This paper has proposed to solve the problem of the validation. Based on the development architecture framework by Struts 2. It can protect the business data effectively. Besides, it is useful to the upper debug and testing. With the development of the web, it is absolutely necessarily for a large scale enterprise to be informational. Custom validation of Struts 2 emphasize particularly on the control of the web page Validation in java. It will also open the new scope and new business opportunities' for the companies and the programmers. It's recommended to use this technology for the Better performance.

REFERENCES

[1] A Saxena,A Chaurasia "Software Project Architectural Approach using Java Struts" International Journal of Research, Vol. 5, Issue 2, 2014 .

[2]. Y Wang , C Guo,L Song."Architecture of E-Commerce Systems Based on J2EE and MVC Pattern"IEEE Computer Society 2009.

[3] R Garg ,Y Sood ,B Kottana, P Totlani "A Framework Based Approach for the Development of Web Based Applications"World of Computer Science and Information Technology Journal (WCSIT)Vol. 1, No. 1, 1-4, Feb. 2011.

[4] P Saxena,S Hanna, P Poosankam, D Song." FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications" In Proc. of the 17th Annual Network and Distributed System Security Symposium (NDSS), Feb 2010

[5] P Gupta,M.C. Govil ."MVC Design Pattern for the multi framework distributed applications using XML, spring and Struts framework" International Journal on Computer Science and Engineering Vol. 02, No. 04, 2010, 1047-1051.

[6] (2009,May) OWASP Top Ten project. [Online]. Available: http://www.owasp.org/index.php/Category: OWASP Top Ten Project

[7] (2009, May) Commons validator. Apache. [Online]. Available: http://commons.apache.org/validator/

[8] (2009, May) Struts. [Online]. Available: http://struts.apache. Org

[9] (2009, September) Hibernate validator. Hibernate. [Online]. Available: https://www.hibernate.org/412.html

[10] L.-H. Netland, Y. Espelid, and K. A. Mughal, "A reflectionbased framework for content validation," in ARES. IEEE Computer Society, 2007, pp. 697–706.

[11] K. Arnold, J. Gosling, and D. Holmes, The Java Programming Language, Fourth Edition. Addison-Wesley, 2006.

[12] F Mancini, D Hovland, K A Mughal," The SHIP Validator: An Annotation-based Content-Validation Framework for Java Applications", 2010 Fifth International Conference on Internet and Web Applications and Services, 2010 IEEE DOI 10.1109/ICIW.2010.26.

[13] D Brown,C M Davis, S Stanlick" Struts in Action" DreamTech, Delhi,India 10,11.

[14] A Saxena" Struts based Approach for the Development of Java Applications" Journal of Software Engineering Tools & Technology Trends Volume 1 issue 2.

[15] A Saxean," Securing Confidential Data using Java/J2EE" IJSTM, Vol. 2 Issue 3, July 2011.