C:\Users\Siva\AppData\Local\Microsoft\Windows\INetCache\Content.Word\zenith 1138.png

**e-Trademine**

**Build and Deployment**

**Document: Build & Deployment**

**Issued on : 25-oct-2015**

**Author : Vinod**

**Version : 1.0**

**Copyright**

ã 2015 Zenith Services Inc.

All rights reserved

The recipient receives this document on the condition that the recipient will keep the document confidential and will not use its contents in any form or by any means without the prior written permission of Zenith Services Inc. Moreover, the recipient is not authorised to place this document at the disposal of any third party without the prior written permission of Zenith Services Inc.

If such permission is granted, it will be subject to the condition that the recipient ensures that any recipient of this document, or information contained therein, is held responsible to Zenith Services Inc. for the confidentiality of that information.

Zenith Services Inc. undertakes to treat all information provided by the recipient as confidential.

The General Terms of Zenith Services Inc. apply to this document.

All trademarks used in this document remain the property of their rightful owners.

**Change of Record**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Release | Change description |
| **25-Jun-2015** | Upendra Gandikota | 1.0 | Initial document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Distribution**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Name | Organization | Comments |
| 1 | Om Nakka | Zenith Services Inc. |  |
| 2 | Neelima Botla | U & I Informatics Pvt Ltd. |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
|  |  |  |  |

E-Trademine.com is an electronic trading and analytical system that allows clients to e-trade any listed instruments, options and provides real-time analysis of the profit and loss analysis of underlying portfolio analysis. It also allows clients do the research and analyze performance of underlying instruments based on historical and real-time data. This application has several integrated modules such real-time streaming of market prices, option pricing and analysis using market standard models such as Black-Scholes model, charting module for real-time and historical analysis.

The Real-time Streaming Tool is a core feature of the e-Trademine application that basically lets users track their favorite stocks simultaneously as the market moves (One minute behind the market moves). It is used for displaying real-time quotes for listed company symbols grouped by the sector per user.

**Technology Stack:**

ExtJS, GSON, RMI Servers, Hibernate and Third party libraries like Fin-cad Libraries for getting real time data based underlying portfolio.

**ExtJS: -**It is used as GUI (Graphical User Interface)

**GSON: -**To Convert JavaScript Objects into Java Objects and as well as reverse fashion.

**RMI Servers: -**RMI’s are used to get some services just like Web Services but in our project we are getting trading service.

**Hibernate: -**You know right why we go for hibernate for interaction with database.

**Fin Cad: -**Fincad is third party Company providing real time data under portfolio. Based on the real time data only we are calculating the risk values in our product.

In Details of Each One:

**ExtJS:-**

Ext JS is a pure JavaScript application framework for building interactive web applications using techniques such as Ajax, DHTML and DOM scripting. In e-Trademine we follow ExtJS MVC Architecture.

ExtJS is a rich user interface JavaScript application framework, it allows us to follow MVC Architecture

In ExtJS we are following Card Layout Architecture means based on button Action the corresponding controller rendering the Desired Screen on to the browser.

**Accordion: -**

This layout manages multiple panels in an expandable accordion style. By default one panel can be expanded at any given time, but you can also expand multiple panels using multi configuration.

**MVC:**

By Using MVC we can separate the logic means model and view and controller readability of the code is improved.

**Model:**

A Model class represents data models. It is used to define the format of data under consideration. It’s similar to a record or row in a table. You can define a Model by giving it a name and listing the fields that the Model is composed of. Let’s define a Model class called Country with fields like name, capital, and population members as shown below.

Ex:-

Ext.define("Country", {

extend: "Ext.data.Model",

fields : ["name","capital","population"]

});

**View:**

 A View is any type of component that is visually represented. For instance, grids, trees and panels are all considered Views.

Ex:-

Ext.define ('MyApp.view.bar.Bar', {

extend: 'Ext.panel.Panel',

xtype: 'bar',

controller: 'bar',

items: [{

xtype: 'foo',

listeners: {

collapse: 'onCollapse',

addrecord: 'onAddRecord'

}

}]

});

**Controller:**

Controllers are used as a place to maintain the view’s logic that makes your app work. This could entail rendering views, routing, instantiating Models, and any other sort of app logic.

Ex:-

Ext.define ('MyApp.view.main.MainController', {

extend: 'Ext.app.ViewController',

requires: [

'Ext.MessageBox' ]

alias: 'controller.main',

onClickButton: function () {

Ext.Msg.confirm('Confirm', 'Are you sure?', 'onConfirm', this);

},

onConfirm: function (choice) {

if (choice === 'yes') { //

}

}

});

**Store:**

A [Store](https://docs.sencha.com/extjs/5.1/5.1.1-apidocs/#!/api/Ext.data.Store) is a client side cache of records (instances of a Model class). Stores provide functions for sorting, filtering and querying the records contained within.This sample application does not contain a store, but not to worry. Simply define your store and assign the Model.

Ex:-

Ext.define('MyApp.store.User', {

extend: 'Ext.data.Store',

model: 'MyApp.model.User',

data : [

{firstName: 'Seth', age: '34'},

{firstName: 'Scott', age: '72'},

{firstName: 'Gary', age: '19'},

{firstName: 'Capybara', age: '208'}

]

});

(OR)

We can write Direct Function for getting the data from DataBase in Store by using the **“directFn”**function provide by DJN Servlet.

Ex:-

Ext.define('MyApp.store.User', {

extend: 'Ext.data.Store',

requires: [

'MyApp.model.User'

],

constructor: **function**(cfg) {

**var** me = **this**;

cfg = cfg || {};

me.callParent([Ext.apply({

autoLoad :**true**,

model :'MyApp.model.User',

storeId: 'Names',

proxy: {

type : 'direct',

directFn : Name of Direct Action.method name,

writer : { type : 'json' },

reader : { type : 'json' }

},

}, cfg)]);

}

});

In e-Trademine the project entire view details are available in E-TrademineView.js and corresponding Controllers having listener or Action logics to send json request to DJN Servlet which handles all requests. In our project Servlet name is Forex Server it handles json request by using @Direct Action Annotation.

**GSON:**

To Convert JavaScript Object (JSON) into Java Object we used GSON libraries provided by Google.

After Converting Json Object into Java Object we send that converted Object into RMI’s.

Ex:-

**publicclass**JSONUtil {

**privatestatic**Gson*gson* = **new**GsonBuilder().create();

**publicstatic** String toJson(Object obj){

**return***gson*.toJson(obj);

}

**publicstatic** Object fromJson(String str, Class<?> clazz){

**return***gson*.fromJson(str, clazz);

}

**publicstatic** Object fromJson(JsonElement element,Class<?> clazz){

**return***gson*.fromJson(element, clazz);

}

}

**RMI Servers:-**

RMI (Remote Method Invocation) is used to provide some services to us just like at runtime it provide an Object instance required by developer.

To Develop RMI Servers We need to follow the below Steps.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmi registry tool
5. Create and start the remote application
6. Create and start the client application

Ex:-

**Step1:-**

**Remote Interface:-**

package com.upendra.service;

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface MyInter extends Remote {

public void wish() throws RemoteException;

}

Note:-The Remote Interface having One abstract method called wish () method. The implementation class is responsible to provide implementation for that method.

**Step 2:-**

**Implementation Class for Remote Interface:-**

/\*\*

\*

\*/

package com.upendra.service;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

/\*\*

\* @author upendra

\*

\*/

public class MyInterfaceImpl extends UnicastRemoteObject implements MyInter {

public MyInterfaceImpl() throws RemoteException {

super();

// TODO Auto-generated constructor stub

}

\* @param args

\*/

@Override

public void wish () throws RemoteException{

System.out.println("Welcome..!! Good Morning");

}

public static void main (String [] args) {

}

}

**Step3:-**

Write RMI Server which binds the Implementation Class Object with PORT\_NUMBER and Alias Name

Ex:-

package com.upendra.server;

import java.net.MalformedURLException;

import java.rmi.AlreadyBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import com.upendra.service.MyConstants;

import com.upendra.service.MyInter;

import com.upendra.service.MyInterfaceImpl;

/\*\*

\* @author upendra

\*

\*/

public class RealServer {

public static void main(String[] args) throws RemoteException, AlreadyBoundException,

MalformedURLException

{

Registry registrey=LocateRegistry.createRegistry(4536);

MyInter obj=new MyInterfaceImpl();

registry.bind(“MyWONSERVER”, obj);

System.out.println("Server Started..!!");

}

}

**Step4:-**

Writing RMI Server Client

Ex:-

package com.upendra.client;

import java.net.MalformedURLException;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import com.upendra.service.MyInter;

import com.upendra.service.MyInterfaceImpl;

/\*\*

\* @author upendra

\*

\*/

public class MyClient {

public static void main(String[] args) throws MalformedURLException, RemoteException,

NotBoundException

{

Registry registry=LocateRegistry.getRegistry(4536);

MyInter inter=(MyInter) registry.lookup("MyWONSERVER");

inter.wish();

}

}

We have 5 Servers are there in e-Trademine they are listed below.

1.QuoteServerImpl

2.HistoricalQuoteServerImpl

3.HistoricalDataServerImpl

4.RealTimeFeederImpl

5.TrademineServer

We need to run these 5 servers before starting our main project called “E-Trademine”. They can be run like the following.

Ex:-

Press Ctrl+Shift+R then type Server name like QuoteServerImpl then press enter it will opened as a Java file run this java file as JavaApplication it will automatically run the QuoteServerImpl.

After run all the above servers then run “E-Trademine” on Server then this main project gets services from RMI Servers.

ForexServer is a Servlet where we configured all RMI Servers details the RMI Servers takes the converted java Object from GSON and send it to corresponding Hibernate Dao’s because Hibernate Dao class Object Configured in RMI Servers,From Hibernate Dao send that Object to Corresponding Hibernate Dao Impl Class whose having Hibernate Logic to store or retrieve the data Logic.

**E-Trademine Project Flow:-**

We are following ExtJS MVC Card Layout right so based on user actions it will take target screen on to the browser.

In ExtJS we have main view available in E-Trademine project under WebContent/app/view/etrademine/EtrademineView.js which is having entire project screens details by following card layout structure.

We have individual controllers in ExtJS side right in that we need to configure view, model, store details to create JSON Object we can get the data from the screen by using getValue() on “id”

Ex:-

var value=Ext.getCmp (“id”).getValue ();

Controller is place where gather all models and views and stores into one place for creating and send the JSON Object.

From the controller we can call ForexServer Servlet Class methods using the following Name Space configured in web.xml.

Ex:-

Ext.Direct.addProvider(zenith.direct.REMOTING\_API);🡪Name Space API Configured in web.xml

forexServer.methodName();

Here forexServer is Action Class alias name in ForexServer just like

@DirectAction(action = "forexServer")

Public Class ForexServer

In the ForexServer Servlet Class we need to convert JSON Object into Java Object right so that we gone through the documentation of GSON which is provided by GOOGLE.

It is a third party libraries we need them in our project lib folder to convert JSON Object into Java Object how to convert JSON Object into Java Object all the details we will find in JSONUtil.java file available under util folder of E-Trademine Project.

By using this class we can easily convert all JavaScript Objects into Java Objects and also in revers fashion it is possible because when the time of getting data from database generally database returns Java objects right we need to convert them into JavaScript Objects to set data to ExtJS Screens right so this one also done through the GSON.

Note: GSON is used to Convert Java Objects to JavaScript Objects and as well as JavaScript Objects into Java Objects.

ForexServer Servlet Class having all sub RMI Servers to provide hibernate model instances and also we configured Server Entry point class details in ForexServer the server reads Services. Properties file in the server entry class based on user requirement it provide instances by taking help of RMI’s.

Before going to run e-Trademine project on server we need some services for trading right so that is the reason we have 5 servers we need to start those servers first to provide services to e-Trademine Project.

The server reads web.xml which is available in e-Trademine project under WEB-INF folder in that we configured DirectJNgineServlet because it will handle all json requests in ForexServer.java file.

From web.xml first it looks for Home.html this is the welcome file within that we configured Etrademine.js file where the ExtJS execution will start here we need to configure first view and corresponding controller in our project they are Login view and Login Controller.

Login Controller having the logic to check whether the input is valid or not given by user. The Login is Success it will search for Forex.html it contains Forex.js this is the main JavaScript file it contains all models and stores and required controller configuration details from here on words the card layout model is enabled.

In front part means ExtJS as User Interface generates JSON requests to send individual screens data to servlet called DirectJNgineServlet in that we configured RMI Servers details so that they are already started right so that we can get target classes instances easily .

For all FxForward, FxSwap, FxOption and FxNDF trading technique is done by using Fincad libraries.

Fin cad is a third party company providing libraries for real time data under portfolio the libraries are developed in C++ we need to convert those services into java

In RMI Servers implementation classes having the hibernate DAO’ instances so that hibernate encapsulates the implementation classes.

Hibernate implementation classes having hibernate transaction logic to save or delete or update logics

**Project Real Flow with Example:-**

Suppose for example if we take User Defaults as an example.

User Defaults is a second button in Admin form of Card layout right in the corresponding controller means app/controller/userdefaults/UserDefaultsController.js file is there it having the logic to generate and transfer the JSON Object to forex server by calling direct method called forexServer.saveUserDefaults() so that JSON Object is sent to ForexServer.java file.

In the ForexServer within the saveUserDefaults() we are calling getUserDefaultData() method to get UserDefaults Model class object by using third party GSON Convertion.

Ex:-

UserDefaultModel userDataModel = (UserDefaultModel) JSONUtil.*fromJson*( data.get(0), UserDefaultModel.**class**);

**return**userDataModel;

finally the getUserDefaultData() method returns UserDefaultModel Object to the caller method called saveUserDefaults() again control comes back to saveUserDefaults() method after that if UserDefaultModel Object is not empty then we are calling tradeServer.saveUserDefaultsData() method here tradeServer is a reference ofTradeDataOperations.

**private** TradeDataOperations tradeServer = ServerEntryPoint.*getTradeServer* ();

Here we are assigning the run time Object getting by calling getTradeServer () on ServerEntryPoint.

ServerEntryPoint is a First Java Class read by application server withing the getTradeServer () of ServerEntryPoint we are calling *server*.getTradeServer () if and only if the object is null (it checks for object in Map first).

Here server is a reference of MainServerOperations(I) it is an interface having common method for those methods all implementations are there in MainServerImpl.java file.

In MainServerImpl we are providing Objects required by user by calling ServiceFactory.getRemoteService(“aliasName”) all alias names configured in Services.properties file based on alias name server get the corresponding Object and put it in Map Object through the Map Object it will return to the caller Method. Here the target Object is TradeDataServerImpl Object because we are passing TradeServer as alias name.

Within the TradeDataServerImpl we have saveUserDefaultsData() method within that we are calling tradeService.saveUserDefaultsData(userDefaultModel) here “tardeService” is a reference of TradeDataService Class within this reference called “tradeService” we are assigning TradeDataService Object by calling static method called TradeDataService.getTradeDataService() within the TradeDataService Class we have saveUserDefaults() within that we are calling Hibernate DAO method so that we are entering into Hibernate through Hibernate DAO Impl classes we are going to save the data or remove the data from the Database.

**Swing Client:**

Swing Client is Standalone e-Trademine Application Developed in Java Swings, Applets and GWT.

If you want to run the Swing Client Open Application Client. Java file then run it as JavaApplication then it is asking Username and Password to Check whether User is Trusted person or not.

Login with proper details then you will get Swing Client User interface.

**UAT Environment:-**

UAT (User Acceptance Test) is Used to Test the Project for this purpose we need to allow users to test our Project without using any java related code so that we need to do the following steps.

UAT Environment Describes Without using any Java Component or Java code The Testers is able to perform Unit Testing and Some Other Tests on the target Project.

Step1:-

Build, deploy and run the RMI servers by using Ant build file which is configure in batch file when double click on batch file then automatically Ant file will run so that all RMI Servers would be run.

Step2:-

Create a WAR file for E-Trademine project and deploy this WAR file into Tomcat Server so that whenever server is started then automatically e-Trademine project is able to run on browser.

If we do above steps then the users doesn’t required any Eclipse to run e-Trademine for Testing Purpose.