Skillshared

THURSDAY, JUNE 6, 2013

Online server console viewer with Ajax long polling.

This post will give you small web application which brings the real time server console to the browser. I am going to explain you, how to run this web application as it is and also if you want to add more customization, how to build the project using the source.

The Comet concept

This application a demonstrative usage of the **Comet** concept introduced by Alex Russell in 2006. Once you get the idea, you can develop innovative applications and make use of it with your web application. For example, Facebook, it shows new friend requests, inbox updates and notifications on time. That is one of the real time usage of **Comet** approach.

The **Comet** is a web application model which always keeps live connection with the server and let the server push the data to the browser through that connection. The browser does not explicitly request specific data. It just keeps live connection with the server. What ever the data pushed (or published) by the server are carried to the browser through that open connection. Also this concept is know as Ajax-Push, Reverse-Ajax, Two-way-web, HTTP Streaming, HTTP-server push.

There are several ways of implementing **Comet** model. The CometD is a java framework developed by Dojo Foundation which implements the **Comet** concept with '**XMLHttpRequest long polling**'. This works with all modern browsers which supports XHR. With XHR long polling, the browser makes an asynchronous request to the server and if there are data available at the server, the response will bring that data to the browser as XML or JSON. After processing the response at client side, browser makes another XHR to poll more available data from the server.

If you have frequent data changes on the server and want to update browsers, the comet approach will be a best fit, because this can be pretty network intensive, and you often make pointless requests, though nothing has happened on the server. For example, frequent stock market information can be published to many users as soon as latest information is available.

How the application works?

This application keeps reading the specified server log file line by line and push collected chunk of lines to the browser as a one string. By default, the application reads the Tomcat's **catalina.out** file. Server runs a timer which is set to every one second and executes these log file reading function. It collects all the lines read within a one second and push to the client. In the client side, application keeps appending new contents to HTML DIV element. The server function will always reads only the newly appending lines of the the log file.

How to run the application?

If you are using Tomcat, you can experiment this application without much worrying. You just need to download the '**console-viewer.war**' file and deploy it in your Tomcat container. If you are testing this with your local configuration, point your browser to following URL.

http://localhost:8080/console-viewer

Click on 'Connect' button and keep looking while you are accessing some deployed application. You will see the Tomcat's console on your web browser. Keep in mind, by default, this application reads '**catalina.out**' file. So you need to start the Tomcat with **./startup.sh** command. If you are appending your application's logs into different log file or using different server, I will explain, how to customize the application for that later.

How to build the project with source?

I have provided complete source to you so that you can download the source code and go one step further or to read different log file or run with different server other then Tomcat. You can build the project using maven. If you are not using maven, you can create your own simple web application and copy the required contents. If you want to get the set of .jar files, you can get it from **console-viewer.war** file.

Here are the steps to build the project using maven.

Download the source and extract it to some where in your local disk.

4 Tweet 9 +1 Widgets

1. Navigate to the location.

cd /home/semika/console-viewer

2. Build the project with the following command.

mvn clean install

You can see, it builds the project and copies the '**console-viewer.war**' file into ***CATALINA_HOME/webapps*** folder.

3. Start the Tomcat with the following command.

./startup.sh

If you want to read some other log file rather than default '**catalina.out**' file, or you want to run this application to view different server's console, you have to provide required log file's absolute path as an initialization parameter of '**Initializer**' servlet as follows and deploy the application on the server.

```
<servlet>
    <servlet-name>initializer</servlet-name>
    <servlet-class>com.semika.cometd.Initializer</servlet-class>
    <init-param>
        <param-name>logFilePath</param-name>
```

POPULAR POSTS

How to create multi module project with maven?
It is more than three years, I have started to use maven and it is a great tool, I have ever used. The support of repository management and...

Don't use 'Query.list()' when you really need 'Query.uniqueResult()' in Hibernate.
I was motivated to discuss a little about this topic, since I am seeing a usage of hibernate which is capable of making server issues in yo...

How to use jQuery grid with struts 2 without plugin ?
When using jQuery with struts 2, the developers are persuaded to use struts2-jQuery plug-in . Because most of the forums and other Intern...

[ERROR] error: error reading *.jar; invalid LOC header (bad signature) - Maven build error
Some time you will get the following error when building a project with maven. Most of the time, this happens after checking out new maven ...

How to use Ajax with CakePHP using jQuery
With this post, I am going to explain how to make AJAX calls using jQuery in cake php . Effective posting related to this topic is very ra ...

BLOG ARCHIVE

- ▼ 2013 (3)
  - ▼ June (1)
    - Online server console viewer with Ajax long pollin...
  - ▶ March (2)
- ▶ 2012 (25)
- ▶ 2011 (6)
- ▶ 2010 (3)
- ▶ 2009 (2)
- ▶ 2008 (11)

skillshared
Follow +1

FOLLOWERS

```
            <param-value>
                /home/semika/apache-tomcat-7.0.37/logs/catalina.out
            </param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
```

Download the application and source

Download console-viewer.war file.
Download source code. If you encounter some issues with this, please send me a mail. I am always ready to give help.

Posted by Semika Loku Kaluge at 5:40 PM 0 comments
Labels: Ajax, CometD, java, jQuery
Location: Colombo, Sri Lanka

FRIDAY, MARCH 29, 2013

Don't use 'Query.list()' when you really need 'Query.uniqueResult()' in Hibernate.

I was motivated to discuss a little about this topic, since I am seeing a usage of hibernate which is capable of making server issues in your application. What I am going to discuss here, is very simple, but some developers are tend to do it in wrong way. Sometime, they are doing this intentionally for safe side.

When we are using hibernate ***Query*** to fetch a single instance, ***Query.uniqueResult()*** is being used except special scenarios. To use ***Query.uniqueResult()***, we must make sure that query will return a single object. But some programmers are still used to do it with ***Query.list()*** method which can pull your application out into very inconsistency situation and issues which are very hard to discover the reason for failure.

Just have a look into the following example.

```
public Account getAccountByAccountIdAndType(Long accountId, AccountType accountType) throws FetchAccountExce

    Account account = null;
    Query query = getSession().getNamedQuery("getAccountByAccountId");
    query.setLong("accountId", accountId);
    query.setString("accountType", AccountType.SAVING.toString());
    List<Account> list = query.list();
    if (null != list && list.size() > 0) {
        account = list.get(0);
    }
    return account;
}
```

The above function is responsible to get Account instance for a given account number and type. Let's assume there can not be two accounts with the same type having same account number.

If you carefully see the above code, the developer has applied many safe factors to minimize the exceptions. Some are even ambiguous. The above method will be functioning well until there is one Account with same type and account number and you expects that too. And also, it will execute well when there are more than one Account with same type and account number, which results wrong output and we don't actually need.

What will happen, if there are more than one accounts with same type and same account number?

That will be a total failure of your application and you will have to compensate if banking system will be on this kind of state. Let's see how the above method will behave in this kind of situation. The above method will hide this critical exception and keeps application flow functioning. The function will get a list which is having the accounts and will return the first Account instance. Ultimately, this will result wrong outputs in your application.

The programmer has added a safe factor to keep application flow functioning even with more than one accounts with same type and number. This is unnecessary and a kind of cheat programming. The programmer can survive for some time, the company who is working for, will have to compensate for loses. This is very poor programming. Some programmers do this intentionally as a safe factor, some are doing this as a quick fix for a defect. But this is very very dangerous.

Further the programmer has checked the "**list**" for ***NULL*** which is ambiguous, because hibernate does not return ***NULL*** list, instead it always returns empty list.

And other shortage of above method is, it may return ***NULL*** as the output which is not a good programming practice. Instead, you can throw an exception.

The **getAccountByAccountIdAndType()** method must return an exception in state. If the programmer used ***Query.uniqueResult()*** for this purpose, it returns the following exception keeping every one enlighten. We can take immediate action to get rid of this killer situation.

org.hibernate.NonUniqueResultException: query did not return a unique result: 2

We can improve the above function as follows.

```
public Account getAccountByAccountIdAndType(Long accountId, AccountType accountType) {

    Query query = getSession().getNamedQuery("getAccountByAccountId");
    query.setLong("accountId", accountId);
    query.setString("accountType", AccountType.SAVING.toString());
    Account account = (Account)Query.uniqueResult();

    if (account == null) {
        throw new RuntimeException("Unable to find Account for account number :" + accountId);
    }
    return account;
}
```

Now the above method will return the exception when there are more than one account with same type and number.

If you want to send more specific error message rather than a hibernate exception message when you are having more than one account with same account number and type, you can put a ***try-catch*** block as follows.

```java
public Account getAccountByAccountIdAndType(Long accountId, AccountType accountType) {

    Account account = null;
    try {
        Query query = getSession().getNamedQuery("getAccountByAccountId");
        query.setLong("accountId", accountId);
        query.setString("accountType", AccountType.SAVING.toString());
        Account account = (Account)Query.uniqueResult();
    } catch(NonUniqueResultException) {
        throw new RuntimeException("Two account found with same account number and type : Acc No-" + accountId
    }
    if (account == null) {
         throw new RuntimeException("Unable to find Account for account number :" + accountId);
    }
    return account;
}
```

As a conclusion, always use ***Query.uniqueResult()*** when you want to fetch a single object with hibernate **Query** and you should make sure your query will return a single object too.

Posted by Semika Loku Kaluge at 1:57 AM 3 comments
Labels: Hibernate, Java Best Practices
Location: Colombo, Sri Lanka

WEDNESDAY, MARCH 20, 2013

How to set browser time to DateTimeItem in smart GWT?

The "**DateTimeItem**" item in Smart GWT allows you to specify the date and the time together as a single field and ultimately it can be mapped to **java.util.Date** field in your model (or a field in your controller class).

```java
DateTimeItem startDateTime = new DateTimeItem("startDateTime","Start Datetime");
```

After picking a date from the calendar, the 'DateTimeItem' shows the value in the following format by default.

MM/dd/yyyy HH:mm

It sets the time part to 00:00. I wanted to have time part to set the browser's time.

After searching on the internet and doing some experiments, I ended up with the following solution.

I am creating a custom class by extending "DateTimeItem" class of smart GWT and overrides the 'transformInput' method.

```java
package mypackage;

import java.util.Date;

import com.google.gwt.i18n.client.DateTimeFormat;
import com.google.gwt.i18n.client.TimeZone;
import com.smartgwt.client.widgets.form.DynamicForm;
import com.smartgwt.client.widgets.form.FormItemInputTransformer;
import com.smartgwt.client.widgets.form.fields.DateTimeItem;
import com.smartgwt.client.widgets.form.fields.FormItem;

public class MyDateTimeItem extends DateTimeItem {

     public MyDateTimeItem(String name, String title) {
        super(name, title);
        this.setUseTextField(true);
        this.setWidth(125);
        this.setInputTransformer(new FormItemInputTransformer() {
            @Override
            public Object transformInput(DynamicForm form, FormItem item, Object value, Object oldValue) {
               DateTimeFormat dtFormet    = DateTimeFormat.getFormat("MM/dd/yyyy HH:mm");
               String strCurrentDateTime  = dtFormet.format(new Date());
               String strSelectedDateTime = dtFormet.format((Date)value, TimeZone.createTimeZone(0));

               if (oldValue != null) { // Changing date.

                   if (strSelectedDateTime.split(" ")[1].equals("00:00")) {
                        //Selecting new date time from the picker while keeping old value in the input field.
                        return strSelectedDateTime.split(" ")[0] + " " + strCurrentDateTime.split(" ")[1];
                   }
                   return strSelectedDateTime;
               }

               return strSelectedDateTime.split(" ")[0] + " " + strCurrentDateTime.split(" ")[1];
            }
        });
    }
}
```

You can use the new component in the similar way in which the '**DateTimeItem**' was used.

```java
MyDateTimeItem startDateTime = new MyDateTimeItem("startDateTime","Start Datetime");
```

Posted by Semika Loku Kaluge at 11:12 PM 1 comments
Labels: Smart GWT

MONDAY, DECEMBER 17, 2012

Spring Controller class for jQuery grid

I have already posted an article of creating jQuery grid using struts 2 action class. This post explains, how to write spring controller class which supports to implement a jQuery grid. If you want to get through of crating a jQuery grid with spring controller class, you must first read my previous post which has shown bellow.

How to use jQuery grid with struts 2 without plugin ?

I am not going to repeat the same stuff which I have explained in my previous post here, because this post has only a few slight differences from my previous post. The previous post used a struts 2 action class as controller and this post uses a spring controller class instead of that. And also, previous post uses **Province** model class directly to represent the data. In this post, I am using a ProvinceDTO (Data Transfer Object) to bring data into controller level. Except above, the whole technique of implementing grid is exactly similar to previous post. So I am 100% sure that, if you go through the previous post first, you will definitely implement the same grid with spring controller class.

When creating jQuery grid with JSON response data, we need to properly synchronize some fields with client side to work grid properly. Those fields include,

- **rows** -The number of rows for one page.
- **page** -The page number requested.
- **sord** -Sorting order.
- **sidx** -Sorting index.
- **total** -Total number of records.
- **records** -The number of records returned for particular search criteria.
- **_search** -Either 'true' or 'false' indicating a particular request is search or not.
- **oper** -The type of operation, either 'add', 'del' or 'edit'

There are a few more fields like above which I have not mentioned here. You can refer jQuery documentation to know those.

In my previous post, with struts 2 action class, I have placed these fields with in the action class itself. With spring controller class, I created separate generic Data Transfer Object to keep these properties and also '**gridModel**' which is a collection of any type holding grid data. Every Data Transfer Object which carries data to create jQuery grid must be inherited from this grid support Data Transfer Object.

The code for grid support Data Transfer Object is as follows.

```
1  package com.blimp.dto;
2
3   import java.util.List;
4
5   public class GridSupportDTO<T> {
6
7   private List<T> gridModel = null;
8
9   private Integer rows = 0;
10  private Integer page = 0;
11  private String sord;
12  private String sidx;
13  private Integer total = 0;
14  private Integer records = 0;
15  private Boolean _search;
16  private String oper = null;
17
18  public List<T> getGridModel() {
19       return gridModel;
20  }
21  public void setGridModel(List<T> gridModel) {
22       this.gridModel = gridModel;
23  }
24  public Integer getRows() {
25       return rows;
26  }
27  public void setRows(Integer rows) {
28       this.rows = rows;
29  }
30  public Integer getPage() {
31       return page;
32  }
33  public void setPage(Integer page) {
34       this.page = page;
35  }
36  public String getSord() {
37       return sord;
38  }
39  public void setSord(String sord) {
40   this.sord = sord;
41  }
42  public String getSidx() {
43       return sidx;
44  }
45  public void setSidx(String sidx) {
46       this.sidx = sidx;
47  }
48  public Integer getTotal() {
49       return total;
50  }
51  public void setTotal(Integer total) {
52       this.total = total;
53  }
54  public Integer getRecords() {
55       return records;
56  }
57  public void setRecords(Integer records) {
58       this.records = records;
59  }
60  public Boolean get_search() {
61       return _search;
62  }
63  public void set_search(Boolean _search) {
64       this._search = _search;
65  }
66  public String getOper() {
67       return oper;
68  }
69  public void setOper(String oper) {
70       this.oper = oper;
71  }
72 }
```

In above class, '**gridModel**' is a generic collection which carries data for the grid. Let's see my **ProvinceDTO.java** class which represent data for each a single grid row.

```java
package com.blimp.dto;

 import java.io.Serializable;

 public class ProvinceDTO extends GridSupportDTO<ProvinceDTO> implements Serializable {

 private static final long serialVersionUID = 88055075892782479 40L;

 private Long id;

 private String name;

 private String provinceStatus;

 public Long getId() {
      return id;
 }

 public void setId(Long id) {
      this.id = id;
 }

 public String getName() {
      return name;
 }

 public void setName(String name) {
      this.name = name;
 }

 public String getProvinceStatus() {
      return provinceStatus;
 }

 public void setProvinceStatus(String provinceStatus) {
      this.provinceStatus = provinceStatus;
 }
}
```

I am going to implement the same grid with same fields as my previous post. In my previous post, I have used **Provice.java** entity class, but here, I am using **ProvinceDTO.java** class, since I don't want to clutter my entity classes with grid specific attributes or don't want to extends from a class which has grid specific attributes. Other thing is, I am using Data Transfer Object pattern with this project.

Next, we'll see our spring controller class which supports to implement jQuery grid.

```java
package com.blimp.webapp.controller;

import java.security.Principal;
import java.util.List;
import java.util.Locale;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import com.blimp.dto.Page;
import com.blimp.dto.ProvinceDTO;
import com.blimp.model.Province;

@Controller
public class ProvinceController {

    @Autowired
    private ProvinceService provinceService;

    @RequestMapping(value = "*search*", method = RequestMethod.GET)
    public @ResponseBody
    ProvinceDTO searchProvince(Principal principal, ProvinceDTO provinceDTO,
    Model model, Locale locale) {

    try {

         Page<Province, ProvinceDTO> requestedPage = new Page<Province, ProvinceDTO>();
         requestedPage.setPage(provinceDTO.getPage());
         requestedPage.setRows(10);

         Page<Province, ProvinceDTO> resultPage = provinceService.findByCriteria(provinceDTO, requestedPage);
         List<ProvinceDTO> provinceList = resultPage.getResultDtoList();

         provinceDTO.setGridModel(provinceList);
         provinceDTO.setRecords(resultPage.getRecords());
         provinceDTO.setTotal(resultPage.getTotals());

    } catch (Exception e) {
         e.printStackTrace();
    }

    return ProvinceDTO;
  }
}
```

Note that, I have passed '**provinceDTO**' as a parameter for above controller method and also the method returns the same '**provinceDTO**' and ultimately as a JSON response to the browser. I have annotated the method with '**@ResponseBody**' annotation which converts what ever the returning stuff from this method into JSON. I have updated '**gridModel**', '**records**' and '**totals**' with appropriate data. The methods of service layer, DAO layer are exactly similar as in my previous post. Don't be lazy, have a look on my previous post, If you really need to implement jQuery grid with spring controller class.

From the client side Javascript code, you need only to change the **'url'** and **'editurl'** attributes of the grid as follows.

url:CONTEXT_ROOT + '/search'
editurl:CONTEXT_ROOT + '/edit'

I haven't implement the controller method for edit function in above controller class. That's it for this post. I hope this will be a good start working with jQuery grid and spring controller. If you have problem of achieving this, feel free to contact me with a simple mail.

Posted by Semika Loku Kaluge at 8:03 PM 0 comments

SUNDAY, NOVEMBER 18, 2012

How to read a dynamically growing file with Java?

This small post explains, how to read a text file which grows dynamically, using Java. The **catalina.out** file which displays tomcat's console is a good example for a file which grows dynamically.

The following example shows, how to read tomcat's **catalina.out** file with Java. This file grows when the applications deployed on tomcat are being accessed by users. The following program displays newly adding contents of **catalina.out** file just one second delay and keep continuing it.

The Java's 'RandomAccessFile' class provides great facilities to read this kind of files. The major usage of it for the following program is, it provides a feature so that we can read a file from a particular point to onwards. Let's see our small program.

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.io.IOException;
4  import java.io.RandomAccessFile;
5  import java.util.Timer;
6  import java.util.TimerTask;
7
8
9  public class ConsoleReader {
10
11    /**
12     * @param args
13     */
14     public static void main(String[] args) {
15     File file = new File("/home/semika/apache-tomcat-7.0.25/logs/catalina.out");
16        try {
17            RandomAccessFile r = new RandomAccessFile(file, "r");
18            //First time read
19            String str = null;
20            while((str = r.readLine()) != null) {
21             System.out.println(str);
22            }
23            r.seek(r.getFilePointer());
24            startTimer(r);
25        } catch (FileNotFoundException e) {
26            e.printStackTrace();
27        } catch (IOException e) {
28            e.printStackTrace();
29        }
30     }
31
32     private static void startTimer(final RandomAccessFile r) {
33      Timer timer = new Timer();
34      timer.scheduleAtFixedRate(new TimerTask() {
35      @Override
36      public void run() {
37      String str = null;
38      try {
39           while((str = r.readLine()) != null) {
40            System.out.println(str);
41           }
42           r.seek(r.getFilePointer());
43      } catch (IOException e) {
44         e.printStackTrace();
45      }
46       }
47   }, 0, 1000);
48     }
49 }
```

In the above program, it reads **catalina.out** file line by line and displays it in console. When you run the program, it will read the whole file up to the end at first and then immediately start a timer to read dynamically adding contents to the file. The timer will always start to read file from a particular point, in above case, it will always start to read the file from the next line which last line was displayed.

While you are accessing your applications deployed in your tomcat container, just keep looking at the output of above program. You can see the tomcat's console just like, you have used to see it before.

Posted by Semika Loku Kaluge at 12:30 AM 0 comments
Labels: java
Location: Colombo, Sri Lanka

Home Older Posts

Subscribe to: Posts (Atom)

RSS

- Online server console viewer with Ajax long polling.
- Don't use 'Query.list()' when you really need 'Query.uniqueResult()' in Hibernate.
- How to set browser time to DateTimeItem in smart GWT?
- Spring Controller class for jQuery grid
- How to read a dynamically growing file with Java?

ATOM

- Online server console viewer with Ajax long polling.
- Don't use 'Query.list()' when you really need 'Query.uniqueResult()' in Hibernate.
- How to set browser time to DateTimeItem in smart GWT?
- Spring Controller class for jQuery grid
- How to read a dynamically growing file with Java?

ABOUT THE AUTHOR

My Photo

Semika Loku Kaluge

Hi,
I am Semika from Colombo, Sri Lanka and working in software engineering field for six years of time by now. Currently, I am working for Shipxpress Inc in Sri Lanka. Primarily, I love to involve with Java developments and related frameworks like Spring, Struts, Hibernate and many more, specially interested in Javascript. I have involved with developments in various domains like Bio-informatics, Hotel Reservation

LABELS

Ajax java Java Best Practices Spring struts 2 tomcat

Management, Stock Markets and Logistics and has provided my service for several companies in Sri Lanka as well as overseas.
One of my hobby is blogging. I have my own blog (skillshared) which shares various tutorials,articles, tips related to software developments.
I am living with my loving wife and little son.

View my complete profile