Generación de código Javascript en Servidor para jtable

Cherencio, Guillermo Rubén
Universidad Nacional de Luján

Abstract
jtable[1] es un plug-in para la librería jquery[2] que permite implementar grillas o las denominadas "crud tables"[3], en el marco del desarrollo de una aplicación web. jtable esta escrito en el lenguaje Javascript y requiere de las librerías jquery y jquery ui[4]; permite la representación de datos en forma de tabla, de manera muy amigable para el usuario. jtable tiene flexibilidad para adaptarse a las distintas necesidades del desarrollador, no obstante, todo el código tiende a extenderse y complejizarse. La visualización de una tabla de datos podría tener cientos de líneas de código Javascript, visibles desde el navegador del usuario final. El presente trabajo pretende resolver la complejidad de las tablas jtable implementando una representación orientada a objetos de la misma del lado del servidor. Los objetos se instanciarán en el contexto del servidor de aplicaciones y estarán mantenidos en memoria, para ser solicitados por la interfaz cliente cuando ésta lo requiera. El presente trabajo permite disminuir la complejidad a partir de una representación más abstracta del problema, obteniéndose además otras ventajas que estimulan nuevos desarrollos y líneas de investigación. Se ha realizado una implementación en lenguaje Java utilizando el servidor de aplicaciones Jetty[5] embebido; el software se distribuye bajo licencia LGPL.

Palabras Clave
jtable jquery AJAX Javascript Java Jetty CRUD table

Introducción

El desarrollo de aplicaciones *n-tier*[1] o de "n-capas" en un ambiente de intranet o internet se ha visto afectado por las nuevas tecnologías que se han desarrollado en los últimos años. El desarrollo de la tecnología AJAX[2] ha permitido una mejora sustancial en la experiencia de los usuarios de este tipo de aplicaciones; los mismos interactúan con la interfaz gráfica como si ésta fuese una aplicación cliente-servidor. Jquery es una librería que fue concebida para uso de AJAX basándose en las capacidades del lenguaje *Javascript* para actuar dinámicamente sobre los elementos de la interfaz del usuario en tiempo de ejecución[3]. Jquery combina llamadas AJAX para intercambiar datos con el servidor de aplicaciones[6]; la interacción del usuario con una interfaz gráfica compleja está apoyada en la librería jquery ui[4] que provee una serie de componentes.

Con estas tecnologías es posible construir interfaces de usuario complejas para *clientes livianos*[7] que permitan una experiencia de usuario más amigable. No obstante, se requiere de una inversión considerable de esfuerzo para combinar todos los componentes y lograr una correcta interacción y ubicación de los mismos en el espacio de trabajo del usuario.

Una forma de mitigar este problema son las aplicaciones basadas en tablas, las cuales, a través de un único componente orientado a datos el usuario puede realizar la mayoría de sus operaciones de forma

1 Ver http://jtable.org
2 Ver https://jquery.org
3 CRUD acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update, Delete)
4 Librería de componentes gráficos para jquery, ver http://jqueryui.com
5 Ver http://www.eclipse.org/jetty
6 El intercambio de datos puede hacerse utilizando mensajes en formato XML o JSON.
7 Se refiere a clientes basados en HTML, el usuario sólo requiere de un navegador web para utilizar la aplicación.

amigable. *jtable* es un plugin para *jquery ui* que permite resolver este problema.

jtable permite crear una tabla en un contenedor de la siguiente forma:

```
$('#contenedor').jtable(arreglo);
```

Donde *contenedor* es una referencia de un contenedor hmtl de la página actual (por ejemplo: un tag); *arreglo* se refiere a un arreglo asociativo[8] *Javascript* que describe la tabla. Para visualizar la tabla, simplemente hay que hacer:

```
$('#contenedor').jtable('load')
```

El problema radica en la complejidad y extensión del arreglo asociativo, dentro de los valores de algunas propiedades se debe incluir código *Javascript*.

El alcance de este trabajo se limita al análisis de todas las propiedades de *jtable*, de su composición, de su representación orientada a objetos, de la generación del arreglo asociativo, del almacenamiento y recuperación de estos objetos dentro del servidor de aplicaciones y de su aplicación en la interfaz gráfica del usuario.

Elementos del Trabajo y metodología

Todo el trabajo se desarrolló en las siguientes etapas, tomando los lineamientos generales de Blanchette[5]:

1. Requerimientos
2. Análisis de propiedades y métodos *jtable*.
3. Diseño e implementación de software.
4. Implementación de tablas de ejemplo y Testing.
5. Empaquetado, publicación y distribución.

1. Requerimientos

El software propuesto deberá implementar una serie de objetos que representen una tabla *jtable* dentro del servidor de aplicaciones, permitirá la creación de objetos *jtable*, los almacenará en memoria y estarán disponibles a solicitud de la interfaz cliente que lo solicite. Los objetos *jtable* permitirán la generación del arreglo asociativo *Javascript* de acuerdo con su estado actual.

2. Análisis de propiedades y métodos *jtable*

Una tabla *jtable* se configura con un arreglo asociativo de propiedades y valores. Las propiedades pueden tener distintos tipos de valores:

1) valor simple: boolean, object, string, number, etc.
2) valor compuesto: arreglo, arreglo asociativo.
3) función: se trata de una función *Javascript* anónima, que puede o no contar con parámetros y posee un cuerpo de código que debemos indicar.

Una tabla *jtable* está formada por los siguientes grupos de propiedades:

1) Generales: 32 propiedades que describen los aspectos generales de la tabla y son aplicables a todas las columnas de la misma.
2) Campos o columnas: 19 propiedades aplicables a cada una de las columnas de la tabla.
3) Acciones: 4 propiedades que describen las operaciones CRUD de la tabla.
4) Métodos: 18 propiedades que describen los distintos métodos que puede invocar

8 Se refiere a un arreglo de la forma { propiedad1 : valor1, propiedad2 : valor2,... }

una tabla (por ejemplo: addRecord(), selectRows(), etc.)
5) Eventos: 13 propiedades que describen los distintos eventos que pueden producirse durante la interacción con el usuario (por ejemplo: rowInserted() ocurre luego de que una tupla se insertó en la tabla)
6) Localización: se trata de un arreglo asociativo que posee la propiedad *messages* y asociado a esta propiedad hay otro arreglo asociativo con 20 pares de propiedad – valor, que indican los distintos mensajes que emite *jtable* al usuario. De esta forma, es posible indicar el idioma de funcionamiento de la tabla.

3. Diseño e implementación de software

El punto anterior planteó algunas decisiones de diseño, pues una misma propiedad podía contener distintos tipos de datos (por ejemplo, la propiedad listAction puede contener una URL o bien el código de una función anónima que devuelva los datos requeridos por *jtable*), una propiedad puede tener un valor por defecto.

Se optó por diseñar la clase SSTable que representa una tabla *jtable*, la cual se compone de los grupos de opciones que se indicaron en el punto anterior. Cada grupo de opciones se representó con una colección de objetos de tipo clave – valor. Cada valor puntual de propiedad se representó con la clase SSOption. La representación de los campos o columnas de la tabla se hizo a través de una colección de objetos de tipo SSField.

A continuación mostramos los diagramas de clases UML resumidos:

SSTable
-name:String -gral:Map<String,SSOption> -fields:List<SSField> -act:Map<String,SSOption> -evt:Map<String,SSOption>
+SSTable() +SSTable(name:String) +SSTable(rs:ResultSet) ...
+generate() ...

Cuando se crea un objeto SSTable, se generan automáticamente todos las propiedades generales con sus valores por defecto. En esta primera versión, se agregan automáticamente los mensajes de localización en español. El constructor a partir de un *ResultSet*[9] fue implementado para crear un objeto SSTable a partir de una consulta sobre una base de datos, a modo de facilitar su creación y prueba. El método generate() permite generar el arreglo asociativo *Javascript* que representa la tabla.

SSField
-name:String -fields:Map<String,SSOption>
+SSField() +SSField(name:String)
+generate() ...

De la misma forma que sucede con SSTable, en SSField se generan todas las propiedades de los campos con sus valores por defecto. El método generate() genera el arreglo asociativo *Javascript* que representa al campo o columna de la tabla.

SSOption
-name:String -dataType:String -defaultValue:String -value:String -mandatory:boolean
+SSOption() +SSOption(name:String) ...
+generate() ...

Debido a la posibilidad de más de un tipo de dato por columna, se optó por

9 Representación del framework Java JDBC de un conjunto de tuplas.

guardar los valores en forma de *String* e indicar el tipo de dato que en este momento tiene la opción. Se asume a toda opción como no obligatoria, mientras que el usuario no cambie su valor intencionalmente; en tal caso, la opción pasa a ser obligatoria. El método generate() genera el arreglo asociativo *Javascript* que representa a esta opción.

4. Implementación de tablas de ejemplo y Testing.

Se desarrolló una aplicación Java, la cual embebe un servidor *Jetty*[10], la misma utiliza una base de datos *Firebird 3.0*[11] y una serie de páginas html que contienen código *jquery* utilizadas para probar el código generado por SSTable.

La forma de prueba es a través de la ejecución del servlet GridServlet asociado a la URL */grid* . GridServlet mantiene en memoria todos los objetos SSTable a probar y reutilizar, a través de un Map de tipo <String,SSTable> en donde el String representa el nombre de la tabla SSTable. GridServlet recibe el parámetro *name* el cual indica el nombre del objeto SSTable requerido. GridServlet devuelve un arreglo asociativo en formato JSON[6] de la siguiente forma, en caso de resultado satisfactorio:

```
{ “Result”:”OK”, “jtable”:{...} }
```

{ … } es un arreglo asociativo *JavaScript* que representa la tabla solicitada. En caso de error, el arreglo devuelto tiene la forma:

```
{ “Result”:”ERROR”, “Message”:”...” }
```

“...” representa el mensaje de error. GridServlet puede ser invocado utilizando el siguiente código jquery:

```
$.ajax({type:'post',datatype: 'json',url:'grid',
  data: {name: 'mesaex'},
  success:function(result){
    var obj = eval(result);
    if  ( obj.Result == "OK" ) {
      $('#queryTable3').jtable(obj.jtable);
      $('#queryTable3').jtable('load');
    } else {
      $("#iretQuery").html("<b>Error!"+
obj.Message+"</b>");
    }
  },error:function(jqXHR,status,errTh) {
    alert('Error!! '+status+' '+errTh);
  }
});
```

El código anterior ejecuta el servlet GridServlet a través de la invocación de la URL /grid, pasa el parámetro name con valor “mesaex”; “mesaex” es una *jtable* mantenida en memoria por GridServlet; si la invocación es satisfactoria, la variable *result* contiene el arreglo asociativo devuelto por GridServlet; se utiliza la propiedad jtable para crear la tabla *jtable* en base al arreglo asociativo generado por GridServlet. Es necesaria la llamada a la función *JavaScript* eval() para que el *string* devuelto por GridServlet se convierta en un arreglo asociativo *Javascript*. El contenedor queryTable3 contendrá la tabla *jtable*, el contenedor iretQuery contendrá el mensaje de error, en caso de que la llamada a GridServlet no haya sido satisfactoria.

El objeto SSTable dentro de GridServlet que es mantenido en memoria vinculado al nombre “mesaex” es el siguiente:

```
SSTable t = new SSTable(“mesaex”);
t.setField("idm:{title:Mesa #}");
t.setField("fecha:{title:Fecha, type:date,
        displayFormat:dd/mm/yy}");
t.setField("codigom:{title:Materia}");
```

10 Jetty es un servidor web y contenedor de servlets de código abierto, disponible en http://www.eclipse.org/jetty

11 Firebird es una base de datos relacional de código abierto, disponible en http://www.firebirdsql.org

```
t.setAction("listAction","JsonServletDates");
mapt.put(t.getName(),t);
```

el objeto t representa en el servidor la tabla *jtable* que se visualizará en el contenedor cliente queryTable3; el objeto mapt es un Map<String,SSTable> que almacena en memoria las instancias de SSTable disponibles en el sistema.

El equivalente de este código en *Javascript*, haciendo todo el trabajo del lado de cliente, sería el siguiente:

```
$("#btn2").click(function(){
  $('#queryTable').jtable({
    title: 'Mesas de Examen',
    defaultDateFormat: 'dd/mm/yy',
    animationsEnabled: true,
    deleteConfirmation: true,
    messages:{
...<20 pares propiedad : valor para mensajes>...
    }
    actions: { listAction: 'JsonServletDates' },
    fields: {
      idm: {  title: 'Mesa #'  },
      fecha: {
        title: 'Fecha',
        type: 'date',
        displayFormat: 'dd/mm/yy'
      },
      codigom: {
        title: 'Materia'
      }
   }
});
  $('#queryTable').jtable('load');
});
```

En este caso, cuando se presiona el boton btn2 se construye en el contenedor queryTable la tabla *jtable*. En el ejemplo se han omitido los 20 mensajes de localización para reducir la extensión del mismo. Puede observarse que se trata de un ejemplo muy trivial, una simple tabla con 3 columnas, sin validaciones, sin listas desplegables u otras tablas relacionadas; en una situación real, la complejidad del código, su extensión y anidamiento, lo convierte en un código menos legible y mantenible. Se trata de 40 lineas de código del lado cliente que pueden reducirse a 5 lineas del lado servidor.

5. Empaquetado, publicación y distribución.

Al momento de escribir este artículo, el software se encuentra aun en etapa de pruebas, generando distintos casos de uso y documentando ejemplos. Se esta trabajando en la generación de documentación para el usuario final, en cuanto a los pasos a seguir en su instalación, luego de la descarga del software. En este momento, las primeras versiones del software pueden descargarse desde http://www.grch.com.ar/docs/soft/jtable.generator/ , favor de descargar y descomprimir; leer las instrucciones en archivo README. El software se distribuye bajo licencia LGPL[12].

Resultados

La primera versión del software propuesto ha demostrado la factibilidad de la solución propuesta al problema planteado.

A través del servlet GridServlet y las tres clases propuestas, es posible crear tablas *jtable* dentro del contenedor Servlet del servidor *Jetty*, mantenerlas en memoria y ser servidas desde allí a todos los clientes que la soliciten.

Los resultados han sido satisfactorios en las primeras pruebas realizadas. Se han agregado métodos para facilitar la creación de los objetos SSTable para evitar que ahora la complejidad se traslade del lado servidor.

Aún falta avanzar con tablas *jtable* anidadas o tablas que tienen relaciones de tipo padre – hijo; así como también con tablas que requieren de paginación.

12 http://www.gnu.org/licenses/lgpl.html

El código realizado no tiene dependencias de terceras librerías, todo el código *Javascript* o *JSON* es generado manualmente por las clases propuestas.

No se observan diferencias en cuanto a su aspecto visual, ni en cuanto a funcionamiento de la interfaz cliente de tablas *jquery* realizadas manualmente con respecto a las descargadas del servidor.

El código cliente se ha simplificado en extensión y complejidad. La complejidad que antes teníamos en el cliente no se ha trasladado al servidor.

Se considera que el proyecto ha logrado alcanzar los objetivos propuestos. Si bien las tablas implementadas son aún de baja complejidad, se espera poder avanzar en ejemplos más sofisticados en los próximos meses.

Discusión

Se espera que la publicación del software permita un *feedback* por parte de usuarios que comiencen a usarlo e incluyan el código en sus nuevos desarrollos.

Se espera que futuros desarrollos permitan la prueba del software propuesto con un número mayor de tablas y una mayor variedad de las mimas.

Las pruebas se han realizado sobre plataforma Linux Debian amd 64 bits utilizando *openjdk* versión 1.8.0_111, se espera que la comunidad de usuarios permita ampliar la prueba a otras plataformas.

Se ha generado un código compacto, teniendo en cuenta la complejidad de la aplicación. Los tiempos de generación de código *Javascript* son aceptables, permiten un trabajo interactivo fluido con el usuario y los requerimientos de hardware son idénticos al requerimiento de Java JRE versión 8. Las pruebas de sucesivas ejecuciones no permiten apreciar una degradación en la performance del sistema.

Desde el punto de vista didáctico, bien podrían hacerse proyectos desde asignaturas tales como: Algoritmos y Estructuras de Datos, Paradigmas de la Programación, Base de Datos, etc.

El desarrollo propuesto presenta las siguientes ventajas:

-Disminución de la complejidad de código a partir de su representación en términos de objetos.
-Generación automática de complejos arreglos asociativos *Javascript* para la creación de tablas *jtables*.
-La generación automática no impide el agregado o cambio de propiedades en el arreglo asociativo en la interfaz cliente.
-La generación de código es compacta, reduce la cantidad de caracteres necesarios para la creación del arreglo asociativo, de esta forma, la cantidad de bytes a transferir desde el servidor al cliente es mínima; sin que ello impacte en la legibilidad del código cliente.
-Permite el reuso de código de tablas y de sus componentes.
-Al mantenerse en memoria principal los objetos SSTable, se obtiene una mayor performance en la creación de las mismas.
-Permite hacer un mejor balance de la carga de trabajo entre cliente y servidor.
-Permite ocultar el código de las tablas *jtable* no siendo visibles desde el cliente.
-No hay posibilidad de error en la construcción del arreglo asociativo *Javascript*, ya que el programador no interactúa directamente con el mismo.
-Facilita la implementación de tablas *jtable* por parte de programadores con escasos conocimientos en *Javascript*.
-En caso de *upgrade* de la librería *jtable*, el impacto en la aplicación queda acotado a

las clases SSTable, SSField, SSOption; sin necesidad de revisar el código de cada una de las tablas de la aplicación.
-Cualquier cambio que se desee sobre el comportamiento de las tablas implementadas en la aplicación es posible hacerlo en forma global y centralizada modificando las clases SSTable, SSField, SSOption.
-Permite un mejor control en cuanto a la comunicación con el servidor, como se indicó en el ejemplo propuesto, es posible tener un contenedor para indicar el error de comunicación con todo detalle, mientras que en el ejemplo manual, la tabla *jtable* se construye sobre el contenedor y ante un error de comunicación simplemente no se visualizará la tabla, sin dar más información al usuario.

Las principales desventajas que se observan derivan del hecho de que aún es una implementación primitiva del software propuesto:

-Aún no se ha desarrollado la posibilidad de implementar tablas relacionadas de tipo padre -hijo.
-Toda nueva tabla que se desee agregar, quitar o modificar implica cambios en el servlet GridServlet, compilación y *deployment* de la aplicación.
-Aún solo disponible para Java.
-Aún no se ha desarrollado la posibilidad de clonar objetos SSTable, SSField, SSOption.

Conclusión

Se ha logrado concebir un software de escasa extensión que provee de múltiples ventajas para el desarrollo de aplicaciones basadas en *jtable*. Los resultados obtenidos estimulan la ampliación del desarrollo hasta alcanzar toda la funcionalidad de *jtable*.

El código fue desarrollado en un entorno de desarrollo BlueJ[13], lo cual facilita su *deployment* y distribución en múltiples plataformas.

Este trabajo abre nuevas lineas de investigación en cuanto a: mejoras que puedan introducirse al software, la creación de nuevos ejemplos de tablas *jtable* que muestren las distintas capacidades que posee, tales como: paginación, ordenamiento, tablas hijas sincronizadas, edición, validación, formato de datos, etc. El presente trabajo puede extenderse a otros lenguajes de programación que permitan la ejecución de código en servidor de aplicaciones.

Agradecimientos
Al Ing. Luis Perna de la Universidad Tecnológica Nacional, Facultad Regional Delta, por facilitarme los medios para la presentación de este trabajo.

Referencias
[1] Sheriff Paul D, Fundamentals of N-Tier Architecture, PDSA, Inc., May 1, 2006.
[2] Dave Crane, Eric Pascarello, Ajax in Action, 2nd edition, MANNING, 2010
[3] Bear Bibeault, Yehuda Katz, jQuery in Action, 2nd edition, MANNING, ISBN 978-1-935182-32-0, 2010
[4] Sarrion Eric, jQuery UI, A Code-Centered Approach to User Interface Design, O'REILLY , ISBN 978-1-449-31699-0, 2012
[5] Blanchette, Jasmin, The Little Manual of API Design, Trolltech, a Nokia company, June 19, 2008, disponible en http://www4.in.tum.de/~blanchet/api-design.pdf
[6] Standard ECMA 404, The JSON Data Interchange Format, 1st Edition, ECMA International, October 2013, disponible en http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Datos de Contacto:

13 Ver http://www.bluej.org/ entorno de desarrollo Java muy simple, utilizado para el aprendizaje del lenguaje.

Cherencio,Guillermo Ruben, Universidad Nacional de Luján, Licenciatura en Sistemas de Información, Ruta 5 y 7, Luján, Pcia. Bs As, Argentina, CP 6700, gcherencio@unlu.edu.ar.