

Primeros pasos con Google Web Toolkit

EL CURRANTE DE LA WEB

El ingenioso Google Web Toolkit es capaz de crear aplicaciones JavaScript optimizadas en tiempo récord. **POR DAN FROST**

He dedicado muchos días, semanas y posiblemente meses de mi vida a JavaScript. El reciente resurgimiento de los entornos de trabajo JavaScript (y su creciente estabilidad) ha ayudado en esto. Google Web Toolkit (GWT) [1] parece ser el próximo nivel en el desarrollo de JavaScript: en lugar de escribir en JavaScript, podemos escribir en Java.

GWT es un entorno para la creación de aplicaciones optimizadas en JavaScript compatibles multinavegador. Con GWT creamos aplicaciones JavaScript programando en Java y compilando el código a un JavaScript altamente optimizado, HTML y CSS. Por mucho que te guste trabajar con los intrincados bugs de un JavaScript multinavegador, llega un momento en que uno

dice basta. En mi caso, GTW llegó justo antes de mi límite.

De Cerca

GWT proporciona una librería de diseños, elementos de formulario y otros componentes para crear aplicaciones Web. En lugar de añadir JavaScript/AJAX encima del HTML y CSS en bruto, podemos usar componentes de alto nivel de Java para que GWT los compile a un JavaScript independiente de navegador, y que probablemente no tengamos ni que depurar.

Sin embargo, esto no implica tener que abandonar por completo la programación en JavaScript. JavaScript aún tiene su lugar. Por eso en este artículo veremos cómo exponer partes de nuestra aplicación generada con GWT en Java-

Script, creando de este modo una API adecuada.

Algunas ventajas de GWT son:

- Compatibilidad de navegadores: si usamos componentes de alto nivel de GWT, no perderemos apenas tiempo en depurar la aplicación.
- Rendimiento: obtenemos código JavaScript y tiempos de descarga optimizados.
- Exposición del API a JavaScript: podemos migrar sección a sección a GWT, haciendo uso del API JavaScript existente.
- Depuración y desarrollo: trabajamos con un lenguaje más robusto que JavaScript, con mejores herramientas de depuración.

En la empresa 3ev hemos creado un CMS basado en front-end que se ubica encima del HTML/CSS de cualquier página Web y permite la edición in situ. Esta aplicación es maravillosa, pero un infierno a la hora de depurar. Se generó primero en JS a medida, luego se migró a Prototype y más recientemente a MooTools, con Ext JS proporcionando algunos elementos Window/Form. Finalmente comenzamos a migrar partes del CMS a GWT. Esta situación es muy común: tenemos una enorme y creciente aplicación en JavaScript que lleva mucho tiempo probar y depurar. GWT parece una solución atractiva, pero ¿cómo podemos utilizarlo en las áreas problemáticas? En este artículo vamos a mostrar cómo usar GWT para generar e integrar componentes JavaScript con las aplicaciones existentes y páginas Web.

Comenzamos

Para comenzar, descargamos una versión adecuada de GWT para nuestro sistema operativo [2]. Esta descarga contiene un conjunto de ejemplos, el compilador de Java a JavaScript y herramientas para crear nuevas aplicaciones y pruebas de ejecución. A continuación, descomprimos el archivo descargado y buscamos el applicationCreator:

```
tar xzf gwt-mac-1.5.2.tar.gz
cd gwt-mac-1.5.2
./applicationCreator -help
```

La herramienta applicationCreator crea un nuevo proyecto GWT para compilar aplicaciones JavaScript. ProjectCreator, ubicado en el mismo directorio, se usa en la creación de proyectos GWT para su edición con Eclipse, pero para este ejemplo vamos a permanecer “agnósticos”.



Para comenzar, creamos un simple proyecto:

```
./applicationCreator -out ➤
~/MiProyectoGwt ➤
com.miempresa.client.MyApp
```

El creador de aplicaciones genera los siguientes archivos en el directorio `~/MyGwtProject`:

```
MyApp-compile
MyApp-shell
src/com/mycompany/
client/MyApp.java
MyApp.gwt.xml
public/
MyApp.css
MyApp.html
```

Este pequeño número de archivos es algo muy adecuado, porque significa que lo que vemos es en lo que estamos trabajando, en lugar de tener directorios y directorios en los que terminaríamos perdidos.

A continuación abrimos `client/myapp.java`, que contiene la clase Java que GWT va a compilar a código JavaScript. Este artículo se limita a una clase, pero podemos reestructurar nuestro código a otras clases a nuestro libre albedrío, en cualquier otro proyecto. Para iniciar la consola de GWT, tecleamos lo siguiente:

```
./MyApp-shell
```

Esto hace que entremos en modo hosted (véase la Figura 1), lo que súbitamente hace de GWT un millón de veces más sorprendente que todo lo que hemos descrito hasta el momento. El modo hosted es un navegador dedicado a nuestro entorno de desarrollo que va a recopilar el código Java a JavaScript cada vez que lo refresquemos. Cuando hacemos un cambio en el archivo Java y refrescamos la consola GWT, veremos los resultados directamente, obviando la necesidad de compilar el código Java cada vez. Mantene-

mos la consola GWT abierta (es decir, el navegador en modo hosted) durante el siguiente paso.

La aplicación que ha creado GWT es rápida y sencilla, mostrando algunos de los componentes estándar de las páginas Web: botones, imágenes, cuadros y cuadros de diálogo. A continuación, abrimos `MyApp.java` y cambiamos

```
dialogBox.setText(i"Bienvenido ➤
a GWT!");
```

a:

```
dialogBox.setText ➤
("Bienvenido a GWT ➤
- ¡¡Esto es asombroso!!");
```

seguidamente refrescamos nuestra consola GWT, pulsamos el botón, y veremos el Java compilado a JavaScript directamente. Por supuesto, el modo hosted sólo es útil durante el desarrollo, por lo que podemos usar `compile` para compilar la aplicación a un conjunto de JS, HTML, CSS y archivos de imagen (Listado 1). La aplicación se compila entonces a un nuevo directorio, `www/com.mycompany.MyApp/`, que contiene los archivos mostrados en el Listado 2.

Abrimos `MyApp.html` con un navegador. Esta aplicación ahora está completamente autocontenida: podemos mover el directorio `www/com.mycompany.MyApp` a cualquier otra ubicación, por ejemplo dentro de la aplicación Web existente:

```
mv www/com.mycompany.MyApp ➤
/ruta/a/mi_antigua_aplicacion/
```

El proceso de trabajar con GWT empieza a tener sentido: en lugar de compilar Java a JavaScript cada vez que terminamos una funcionalidad o arreglamos un bug, podemos trabajar en modo hosted hasta que esté

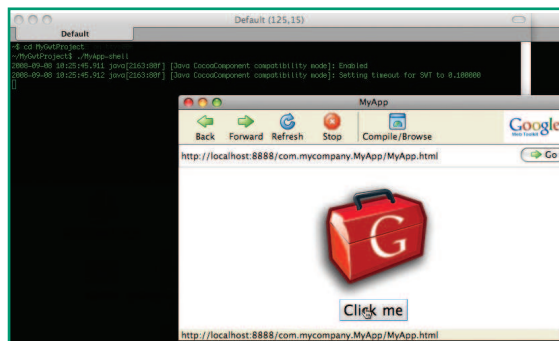


Figura 1: Entramos en el modo hosted de GWT.

completo al 90% y entonces compilarlo a JavaScript.

Dentro de Java

La clase Java implementa `EntryPoint/onModuleLoad`, la cual dispara GWT en cada carga de página, por lo que podemos pensar en ella como el `onload` de las páginas HTML o el `dom ready` de librerías como MooTools. Un pequeño ejemplo muestra esto: editamos `MyApp.java` de manera que la clase contenga sólo las líneas mostradas en el Listado 3.

De forma intuitiva, `Window.alert()` está llamando a la función de alerta de JavaScript. Si activamos el modo hosted con `MyApp-shell`, observaremos que no sucede nada, pero estamos comenzando a ver cómo la experiencia de usuario de JavaScript se ajusta en torno al código Java.

Ejemplo de Cuadro de Diálogo

A continuación vamos a generar un sencillo cuadro de diálogo que se carga desde una URL AJAX existente. Nos aseguramos de que estamos importando todo desde el paquete cliente de GWT:

Listado 2: Contenido de com.mycompany.MyApp

```
01
548CDF11D6FE9011F3447CA200D7FB
7F.cache.png
02
9DA92932034707C17CFF15F95086D5
3F.cache.png
03
A84A8EF7341E8139F58DC5FC2AD52F
22.cache.html
04 MyApp.css
05 MyApp.html
06 clear.cache.gif
07 com.mycompany.MyApp.nocache.js
08 gwt/history.html hosted.html
```

Listado 1: Compilar con GWT

```
01 # ./MyApp-compile
02 Compiling module com.mycompany.MyApp
03 2008-08-30 15:14:35.774 java[2748:80f] [Java CocoaComponent
compatibility mode]: Enabled
04 2008-08-30 15:14:35.775 java[2748:80f] [Java CocoaComponent
compatibility mode]: Setting timeout for SWT to 0.100000
05 Compilation succeeded
06 Linking compilation into ./www/com.mycompany.MyApp
```

Listado 3: MyApp.java

```
01 public class MyApp implements
    EntryPoint {
02     //This is the entry
    point method.
03     public void
    onModuleLoad() {
04
    Window.alert("Hello");
05     }
06 }
```

```
import
com.google.gwt.user.client.*;
```

Reemplazamos el contenido de *onModuleLoad* con:

```
openDialogBox();
```

y añadimos la función mostrada en el Listado 4.

Si abrimos esto en modo hosted, el cuadro de diálogo aparece cuando se abre la página, lo que es una buena demostración, aunque no sea verdaderamente útil. En su lugar, queremos ser capaces de llamar al cuadro de diálogo desde cualquier parte de la aplicación, y para hacerlo, tenemos que exponer parte de la aplicación Java a JavaScript utilizando el JavaScript Native Interface (JSNI).

En primer lugar creamos una función que se declare a sí misma "nativa" e inicie de forma efectiva un API JavaScript, como la función mostrada en el Listado 5.

Por último, en *onModuleLoad*, reemplazamos *openDialogBox()* por *initJavaScriptAPI()*:

```
initJavaScriptAPI(this);
```

Listado 4: Añadir esta Función

```
01 public void openDialogBox() {
02     final DialogBox dialogBox = new DialogBox();
03     dialogBox.setText("This is my simple dialog box");
04     dialogBox.setAnimationEnabled(true);
05     dialogBox.center();
06 dialogBox.show();
07 }
```

Listado 5: Pasar a Nativo

```
01 private native void initJavaScriptAPI (MyApp myapp) /*-{
02     $wnd.openDialog = function () {
03         myapp.@com.mycompany.client.MyApp:openDialogBox();
04     };
05 }-*/;
```

Si refrescamos el modo host, no veremos nada, ya que cuando la aplicación está cargada, sólo declara la API JS: únicamente se abre una ventana cuando se llama a la función JavaScript *openDialog()*. Para ver cómo funciona esto, añadimos la siguiente línea dentro de *<body>* en *public/MyApp.html*:

```
<a href=
"javascript:openDialog();"
">Abrir cuadro de diálogo
GWT</a>
```

A continuación, refrescamos el modo host, pulsamos sobre el enlace, y veremos cómo se abre el cuadro de diálogo. Aunque este es un ejemplo básico, demuestra algo útil: ahora podemos programar funcionalidades complejas, multinavegador con GWT y llamar a estas funcionalidades desde nuestras aplicaciones JavaScript existentes.

Uso de Aplicaciones AJAX Existentes

El cuadro de diálogo es simplemente una sencilla demostración, en lugar de algo verdaderamente útil, por lo que vamos a animarla un poquito añadiendo AJAX entre un servidor AJAX ya existente y GWT.

Un requerimiento usual de un cuadro de diálogo es cargar su contenido sobre AJAX, cosa que podemos conseguir fácilmente modificando nuestra clase. Añadimos el código del Listado 6 al final de *openDialogBox()*.

A continuación, usamos *MyApp-compile* para desplegarla en una aplicación existente: necesitaremos algunas páginas Web en funcionamiento de forma local. Vamos a suponer que trabajamos sobre un sistema LAMP.

Compilamos la aplicación y copiamos el directorio *www* a nuestra aplicación existente:

```
./MyApp-compile
mv -r www
/ruta/a/mi/app/gwt-www
touch /ruta/a/mi/app/gwt-www/
AjaxServer.php
```

finalmente, creamos un archivo llamado *AjaxServer.php* y añadimos lo siguiente:

```
<php
echo "Hola Mundo desde mi
servidor AJAX llamado desde
GWT, lanzado desde una llamada
nativa JavaScript.";
?>
```

Probamos la Nueva Funcionalidad

Para probar la nueva funcionalidad AJAX, abrimos *MyApp.htm* desde dentro de la aplicación y pulsamos sobre el enlace (véase la Figura 2). El API JavaScript significa que podemos llamar a funcionalidades GWT

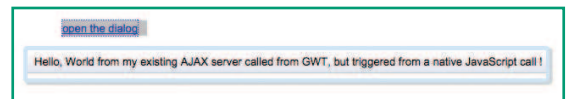


Figura 2: Verificamos *MyApp.htm* para saludar al mundo.

desde una aplicación JS existente, y el uso del servidor AJAX significa que GWT se puede integrar con nuestra funcionalidad AJAX existente. Sin embargo, la URL AJAX es fija, por lo que llevamos esta URL a una variable que se pasa desde JavaScript hasta Java.

En primer lugar, modificamos los contenidos de *initJavaScriptAPI* (véase el Listado 7).

Alternativas

Una alternativa a usar GWT es utilizar uno de los entornos de trabajo JavaScript, como *Scriptaculous/prototype*, *MooTools*, *jQuery* o *Ext JS*.

Estos entornos de trabajo son extremadamente robustos e importan nuestro JavaScript, pero no hacen lo mismo que GWT. Para fragmentos menores (no pequeños) y específicos de AJAX/Web 2.0, estas librerías son magníficas, pero después de un tiempo, depurar, mantener y optimizar un entorno puro JavaScript lleva demasiado tiempo.

El argumento *url* de la primera línea es un parámetro JavaScript, que será convertido a una variable Java de tipo *java.lang.string* y se pasará a *openDialogBox*.

Luego modificamos *openDialogBox* para aceptar el argumento

```
public void
openDialogBox(String url) {
```

y modificamos la petición para usar esta variable:

```
RequestBuilder builder =
new RequestBuilder
(RequestBuilder.GET,
URL.encode( url ));
```

La compilamos y movemos los archivos hasta nuestra aplicación, luego añadimos algunas URLs a las llamadas de la función JS de manera que podamos llamar a las URLs existentes:

```
<a href=!javascript:openDialog
('/ AjaxServer1.php');">
Open my GWT dialog box</ a>
<a href="javascript:openDialog
('/ AjaxServer2.php');">
Open another dialog box</ a>
```

Como paso final, obtendremos la funcionalidad GWT en la aplicación Web con la inclusión de un archivo JavaScript. Añadimos la etiqueta script a la cabecera HTML, ajustando *src* = "." para que apunte al directorio adecuado:

```
<script type="text/javascript"
language="javascript" src=
"com.mycompany.
MyApp.nocache.js"></script>
```

a continuación, en alguna parte de nuestra plantilla, añadimos enlaces que disparen el JavaScript:

```
<a href="javascript:openDialog
('/ruta/a/ajax.php');">
Abrir mi servidor Ajax</a>
```

A modo de prueba, podríamos poner estos enlaces en WordPress, nuestro CMS o cualquier otra página Web.

Optimizar Código

Mi experiencia es que el código JavaScript se hace poco manejable cuando la aplicación crece mucho. Particularmente, paso mucho tiempo optimizando código a bajo nivel. Del mismo modo, debido a que JavaScript es un lenguaje de scripting y débilmente tipado, a menudo nos costará encontrar los bugs.

Migración

Pasar del ligero JavaScript al robusto y pesado Java no es trivial, pero es posible migrar lentamente o simplemente migrar aquellas áreas problemáticas de la aplicación. Para migrar funcionalidades JavaScript existentes a GWT, debemos comenzar con una API bastante sólida. Generalmente, esto significa que estamos llamando sólo a una o dos funciones de nuestra página Web.

Podemos comenzar generando los componentes y las funcionalidades con GWT mediante el uso de sus componentes y funcionalidades, en lugar de HTML y JavaScript en bruto. Es importante hacer uso de componentes de alto nivel que proporcionen seguridad multinavegador.

A continuación, exponemos partes específicas del componente GWT a JavaScript con el uso de funciones "nativas". Estas funciones probablemente serán parecidas a nuestra API existente, de manera que se mantenga la compatibilidad. Finalmente, sólo tenemos que incluir el JavaScript generado con GWT y eliminar el viejo JavaScript nativo de nuestra página Web. ■

Listado 6: Añadir AJAX

```
01 RequestBuilder builder = new RequestBuilder(RequestBuilder.GET,
URL.encode("/AjaxServer.php"));
02
03 try {
04 Request request = builder.sendRequest(null,
05 new RequestCallback() {
06 public void onError(Request request, Throwable exception)
07 // Couldn't connect to server (could be timeout,SOP
violation, etc.)
08 dialogBox.setText( "Sorry - Could not load the HTML");
09 }
10
11 public void onResponseReceived(Request request, Response
response) {
12 if (200 == response.getStatusCode()) {
13 dialogBox.setText( response.getText() );
14 } else {
15 // Handle the error. Can get the status text from
response.getStatusText()
16 dialogBox.setText( "Sorry - I got the response, but don't
understand it!");
17 }
18 }
19 }
20 );
21 } catch(Exception e) { }
```

Listado 7: Modificar initJavaScriptAPI

```
01 $wnd.openDialog = function (url) {
02
myapp.@com.mycompany.client.MyApp::openDialogBox(Ljava/lang/String);(
url);
03 };
```

RECURSOS

- [1] Google Web Toolkit: <http://code.google.com/webtoolkit/>
- [2] Descarga de GWT: <http://code.google.com/webtoolkit/download.html>
- [3] Gwt-fx (animación básica para GWT): <http://code.google.com/p/gwt-fx/>
- [4] GWT Ext JS (entorno de trabajo para GWT): <http://extjs.com/products/gxt/>
- [5] GWT on Rails: <http://code.google.com/p/gwt-on-rails/>