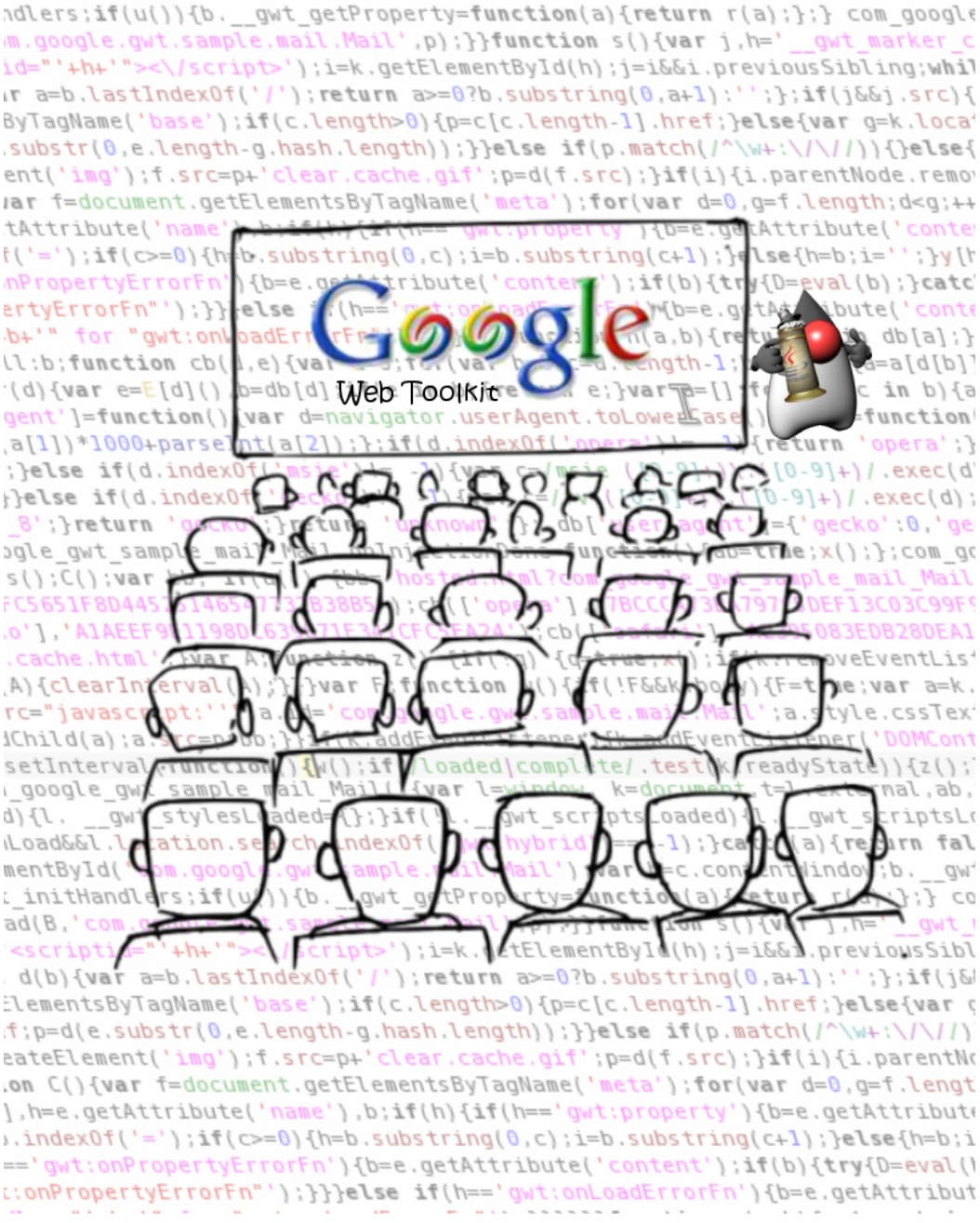
Tutorial (1.0) de Introducción a Google Web Toolkit



por Cristian Castiblanco | casidiablo@elhacker.net

Indice

1. Visión general de Google Web Toolkit

Al parecer estás interesado en conocer acerca de Google Web Toolkit, talvez seas un desarrollador de software o talvez simplemente tengas curiosidad, en todo caso aquí podrás encontrar bastante información al respecto, conociendo lo fundamental. Este tutorial fue basado, en gran parte, en la documentación oficial del proyecto Google Web Toolkit que puedes encontrar visitando éste enlace.

1.2 ¿Qué es Google Web Toolkit?

Google Web Toolkit (GWT) es un framework de desarrollo en Java de código abierto, que te permite escapar de la "matriz" de tecnologías usadas actualmente para escribir aplicaciones AJAX, las cuales son difíciles de manejar y propensas a errores. Con GWT, puedes desarrollar y depurar aplicaciones AJAX usando el lenguaje de programación Java en el entorno de desarrollo de tu preferencia (me refiero al sistema operativo y a los IDEs). Cuando haz acabado tu aplicación (que la has escrito en Java), GWT compila y traduce dicho programa a JavaScript y HTML compatible con cualquier navegador web.

Éste es el ciclo de Desarrollo de GWT:

1. Usa tu entorno de desarrollo integrado (IDE) favorito para escribir y depurar una aplicación en Java, usando las librerías GWT que necesites.
2. Usa el compilador de Java a JavaScript de GWT para transformar tu aplicación en un conjunto de archive JavaScript y HTML que puedes colgar en cualquier servidor y ejecutar desde un navegador web.
3. Verifica que tus aplicaciones trabajan sobre todos y cada uno de los navegadores que consideres que tus clientes usarán.

1.3 Características de Google Web Toolkit

Componentes de la interfaz de usuario dinámicos y re-utilizables

Crea un Widget para construir otros. Coloca los Widgets automáticamente en Paneles. Envía tus Widget a otros desarrolladores en archivos JAR.

RPC realmente fácil

Para comunicarte desde el navegador que lanza tu aplicación con tu servidor web, solamente necesitas definir clases de Java serializables para las peticiones y respuestas. En producción, GWT serializa automáticamente las peticiones del navegador y de-serializa las repuestas desde el servidor web. El mecanismo de RPC de GWT puede incluso manejar jerarquía de polimorfismo en clases, y puedes manejar las posibles excepciones.

Administración del historial del navegador

Las aplicaciones en AJAX no necesitan utilizar el botón "atrás" (back) del navegador. Y GWT no es la excepción, es decir, no es necesario que llames a otras páginas para realizar las diferentes acciones, ni recargar el navegador ni nada.

Depuración en tiempo real

Para cuando tu aplicación esté lista, el código de la misma es traducido a JavaScript, pero mientras lo estás desarrollando este corre sobre una Java virtual machina (JVM). Lo que significa que en la fase de Desarrollo tienes la posibilidad de depurar tu aplicación con los avanzados sistemas de debugging y manipulación de excepciones incluidos en IDEs como Eclipse.

Compatibilidad con los navegadores

Tus aplicaciones en GWT serán automáticamente soportadas por navegadores como FireFox, Internet Explorer, Mozilla, Safari, y Opera sin ningún tipo de operación para la detección de los mismos, en la mayoría de los casos.

Integración con Junit

Mediante la integración de JUnit en GWT tu puedes probar tus aplicaciones y depurarlas en un navegador mientras las construyes...incluso, puedes testear llamadas asíncronas a procedimientos remotos RPC.

Internacionalización
Crea aplicaciones y librerías de Internacionalización rápida y fácilmente.

Interoperability and fine-grained control

Si las librerías de clases de GWT no son suficientes para lo que necesitas, puedes mezclar JavaScript en el código de tu aplicación usando la interfaz nativa de scripts de Java (JavaScript Native Interface, JSNI).

GWT es un proyecto de código abierto

Todo el código de GWT está disponible bajo la licencia Apache 2.0. Si estás interesado en contribuir, por favor visita esta pagina.

Si necesitas una guía de instalación y uso paso-a-paso, puedes saltar a ésta sección.

1.4 ¿Porqué traducir código Java a JavaScript?

La tecnología Java ofrece una plataforma de desarrollo productiva, y con GWT, se puede convertir en una plataforma sólida para el desarrollo de tus aplicaciones AJAX. Aquí están algunos de los beneficios de desarrollar con GWT:

- Puedes usar cualquiera de tus IDEs favoritos (Eclipse, IntelliJ, JProfiler, JUnit).
- Los errores comunes en JavaScript (errores de sintaxis, por ejemplo) son fácilmente detectados mientras desarrollas la aplicación, y no cuando el usuario final lo esté ejecutando.
- El "refactoring" automático en Java está muy de moda en estos días.
- Los diseños en Java basados en la programación orientada a objetos es fácil de comunicar y entender, por ende hace la base de tu código AJAX más comprensible con menos documentación.

1.5 Usando Google Web Toolkit

En GWT puedes usar componentes de interfaz de usuario llamados Widgets, para construir aplicaciones AJAX con GUIs atractivas. Al igual que en la mayoría de los lenguajes de programación, los componentes de la UI son agrupados en paneles que determinan la ubicación de los mismos. A continuación veamos una completa aplicación que utiliza un botón y un manejador de eventos:

```
public class Hola implements EntryPoint

{
 public void onModuleLoad()

 {
  Button b = new Button("Chuzame", new ClickListener()

  {
   public void onClick(Widget sender)

   {
    Window.alert("Hola, geek");
   }
  });
  RootPanel.get().add(b);
 }
}
```

GWT soporta una gran cantidad de widgets que son útiles en el desarrollos de aplicaciones AJAX, incluyendo árboles, pestañas, barras de menú y menús de dialogo. GWT también soporta invocación de métodos remotos (RPC) y otras características. Ver la lista de características para más información.

1.6 Depuración y desarrollo de aplicaciones en GWT

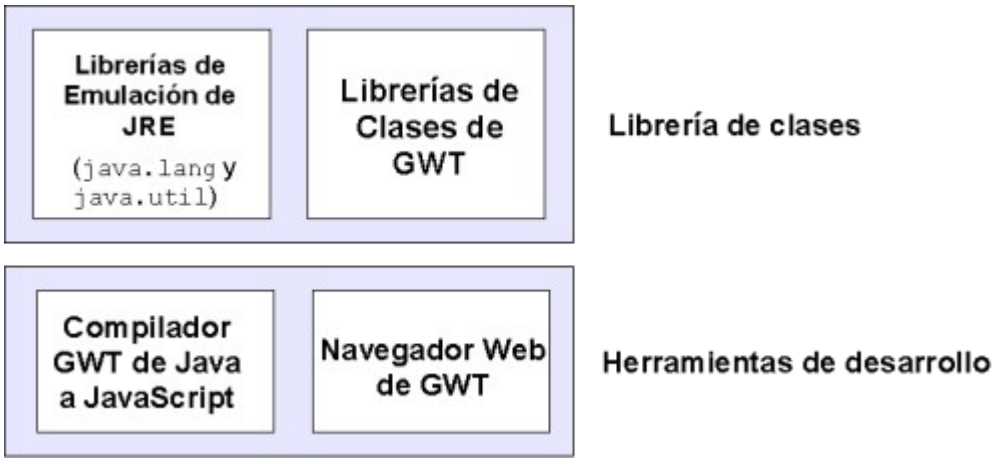
Las aplicaciones GWT pueden ser ejecutadas en dos modos:

- **Modo hosted (hosted mode)** – En modo hosted, tu aplicación corre como bytecodes de Java sobre una máquina virtual. Por lo general gastarás más tiempo desarrollando en modo hosted, ya que allí cuentas con todas las ventajas que te proporciona Java para depurar usando un IDE como Eclipse.
- **Modo Web (Web mode)** – En mode web, tu aplicación corre como HTML + JavaScript sobre un navegador, traducido desde tú código fuente Java original con el compilador de GWT (Java-to-JavaScript compiler). Cuando tu aplicación está terminada, lo único que debes hacer es subirla a un servidor web, y los usuarios finales accederán a ella a través de un navegador en "modo web".

Para soportar el modo hosted, GWT cuenta con un navegador especial que está "enganchado" a la máquina virtual de Java. Ver el diagrama de arquitectura GWT para más información.

1.7 La arquitectura de Google Web Toolkit

GWT tiene cuatro componentes principales: un compilador Java-a-JavaScript, un navegador web "hosted", y dos librerías de clases:



Los componentes son:

- **Compilador GWT Java-a-JavaScript**
  El Compilador GWT Java-a-JavaScript traduce del lenguaje de programación Java a JavaScript. El compilador se utiliza cuando necesites correr tu aplicación en modo web.
- **Navegador web "Hosted" de GWT**
  El Navegador web "Hosted" de GWT te permite correr y ejecutar GWT aplicaciones en modo hosted, donde lo que estás corriendo son bytecodes de Java sobre una máquina virtual sin compilarlos a JavaScript. Para lograr esto, el navegador GWT incrusta un controlador de browser especial (un control del Internet Explorer sobre Windows o un control de Gecko/Mozilla sobre Linux) con hooks dentro de la máquina virtual de Java.
- **Emulación de librerías JRE**
  GWT contiene implementaciones en JavaScript de las librerías de clases más usadas en Java, incluyendo la mayoría de las clases del paquete java.lang y un subconjunto de clases del paquete java.util. El resto del estándar de librerías de Java no es soportado nativamente con GWT. Por ejemplo, las clases de los paquetes como java.io no se utilizan en aplicaciones web ya que estas acceden a recursos en la red y al sistema de archivos local.
- **Librería de clases de interfaz de usuario de GWT**

  Las librerías de clases de interfaz de usuario de GWT son un conjunto de interfaces y clases personalizadas que te permiten crear "widgets" para el navegador, como botones, cajas de texto, imágenes, y texto. Éste es el núcleo de las librerías de interfaz de usuario para crear aplicaciones GWT.

Si tienes curiosidad de entender el código de GWT, por favor siéntete libre de ver esto. El código fuente de GWT está disponible bajo la licencia Apache 2.0.

2. Comenzando en Google Web Toolkit

Las siguientes entradas te proporcionarán la información suficiente, para que des tus primeros pasos en el desarrollo bajo Google Web Toolkit.

2.2 Instalación de Google Web Toolkit

Lo primero es descargar el archivo adecuado de acuerdo al sistema operativo que uses:

http://code.google.com/webtoolkit/download.html

Instalación sobre Windows

Luego de descargar la versión de GWT para Windows, descomprime del archivo y añade su ruta en el path de tu sistema operativo. Para ello, das clic derecho en "*Mi Pc" -> Propiedades -> Variables de entorno,* y en *Variables del sistema* le das doble clic a Path, ya añades la ruta donde descomprimiste el GWT, así:

Ten cuidado de no borrar las rutas ya presentes, ya que esto puede volver inestable tu sistema. También recuerda que las rutas son separadas con punto y coma (;), tal como se vé en la imagen.

Esto se hace para tu comodidad a la hora de desarrollar, ya que de esta manera no es necesario que te situes en el path de instalación, desde la línea de comandos, para ejecutarlos comandos más comunes.

Instalación sobre sistemas operativos GNU/Linux

Luego de descargar el archivo correspiente para GNU/Linux, lo descomprimes con tu gestor de archivos comprimidos preferido, así:

```
tar xvzf gwt-linux-1.4.60.tar.bz2
```

Luego es necesario añadir la ruta donde descomprimiste los archivos al PATH de tu sistema. Para ello debemos editar el archivo `.bashrc` de tu sesión:

```
echo "#Path GWT" >> ~/.bashrc
echo "PATH=$PATH:/ruta/gwt" >> ~/.bashrc
echo "export PATH" >> ~/.bashrc
source ~/.bashrc
```

Hacemos esto para añadir la ruta donde se encuentra GWT a la variable de entorno PATH de tu sistema, con lo que podremos ejecutar los programas desde la línea de comandos, sin necesidad de situar la misma sobre la ruta del GWT.

2.3 Crear una aplicación de cero

Si estás leyendo esto imagino que ya has instalado el Google Web Toolkit en tu sistema operativo, y habrás configurado adecuadamente las variables de entorno.

Google Web Toolkit viene con una utilidad (para consola/línea de comandos) incluida llamada `applicationCreator` que genera automáticamente los archivos que necesitarás para comenzar a desarrollar un proyecto GWT. Además puede también generar proyectos para el IDE Eclipse lo que permite que el desarrollo y depuración de aplicaciones GWT sea bastante fácil.

Crear una nueva aplicación

Basado en la estructura básica de un proyecto GWT, la clase principal de tu aplicación debe estar en un subpaquete de `client`. Puedes crear una nueva aplicación llamada HolaGWT de la siguiente manera:

```
applicationCreator com.loquesea.client.HolaGWT
```

El script `applicationCreator` generará un número de archivos en `src/com/loquesea/`, incluyendo el típico "Hello, world" en la clase `src/com/loquesea/client/HolaGWT.java`. Además, el script genera otro script llamado `HolaGWT-shell` que sirve para correr nuestra aplicación en modo hosted, y un script para compilar la aplicación a JavaScript llamado `HolaGWT-compile`.

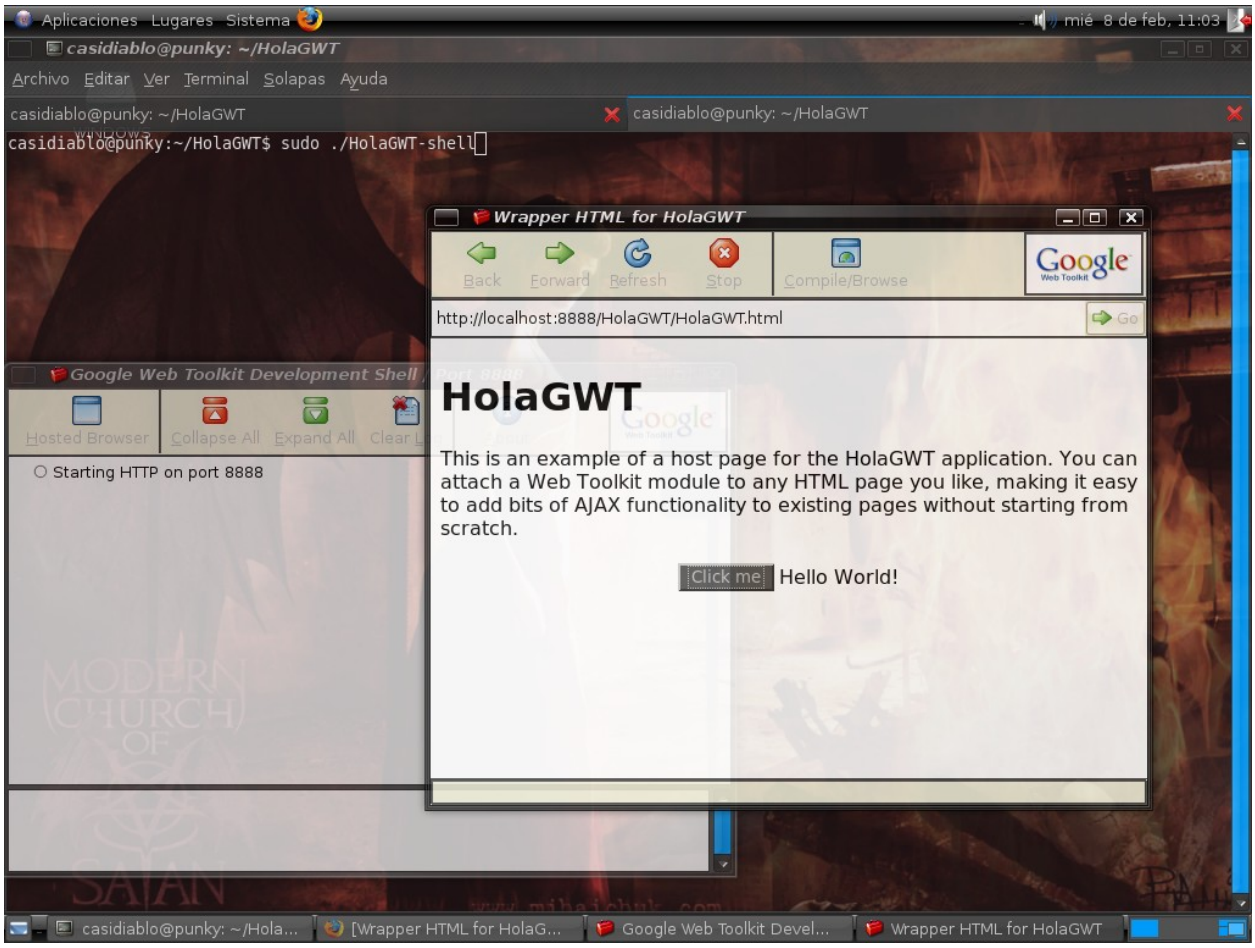
No es necesario indicar un nombre de proyecto tan largo, esto solo se hace para "organizar" mejor las cosas. Lo único obligatorio es la creación de paquete client, por lo que si quieres algo simple te bastará con ejecutar `applicationCreator client.MiAplicacion`.

> ***Para tener en cuenta...*** *en GNU/Linux los script mencionados anteriormente no llevan extensión, y además es necesario ejecutarlos con privilegios de administración (en algunos casos). Puedes usar el comando `su`, para logarte como root. O si usa Ubuntu puedes anteponer el comando `sudo`.*

En Windows, los scripts tienen la extensión .cmd, y su utilización es la misma. En este caso, ya que uso GNU/Linux, el ejemplo va a ser sobre Debian.

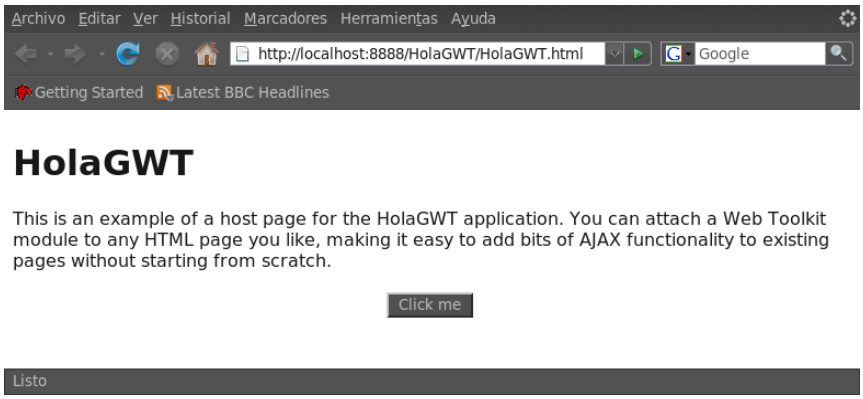
Ejecuta tu nueva aplicación en modo hosted

Para correr tu nueva aplicación en modo hosted, ejecuta el script `HolaGWT-shell`:



Ejecuta tu nueva aplicación en modo web

Puedes también usar el botón "*Compile/Browse*" del navegador, con lo cual GWT compilará automáticamente tu aplicación a JavaScript, y lanzará la aplicación en tu navegador Web predeterminado.



Para los amantes del pingüino... en ocasiones, en sistemas GNU/Linux, es necesario configurar la variable de entorno GWT_DEFAULT_BROWSER, en donde debes colocar la ruta de tu navegador preferido. Puedes hacer esto así:

```
echo "#Path navegador predeterminado para GWT" >> ~/.bashrc
echo "GWT_DEFAULT_BROWSER=/usr/bin/firefox" >> ~/.bashrc
echo "export GWT_DEFAULT_BROWSER" >> ~/.bashrc
source ~/.bashrc
```

Compilar tu aplicación

El proceso de compilación, traduce todo el código en Java que hayas escrito y lo traduce a JavaScript. Para compilarlo solamente debes ejecutar el script `HolaGWT-compile`. Cuando lo hagas por primera vez, se creará un directorio llamado www, en donde se encontrarán todos los archivos HTML y JavaScript de tu aplicación. De esta forma, y cuando hayas terminado la aplicación, simplemente tendrás que copiar dichos archivos al servidor web de producción, y estará todo listo para que un usuario final disfrute de tu programa.

Modifica la aplicación por defecto

Como ya hemos dicho, cuando creas una nueva aplicación, GWT te crea por defecto un archivo `.java` con el nombre de tu aplicación, y dentro de este un ejemplo de un "Hola mundo", por lo que para crear tu aplicación solo debes modificar dicho archivo.

Ésta entrada es solo una pequeña introducción, por lo que nos limitaremos a hacer solo lo ya comentado. Para ver ejemplos de creación de aplicaciones un más avanzadas, puedes ver los ejemplos que vienen incluidos en la carpeta de instalación de Google Web Toolkit.

2.4 Creando una aplicación con Eclipse

Con GWT puedes crear proyectos de Eclipse que te permitirán usar este magnífico IDE para desarrollar y depurar tu aplicación. Para crear un proyecto de Eclipse es necesario primero crear una aplicación base, así:

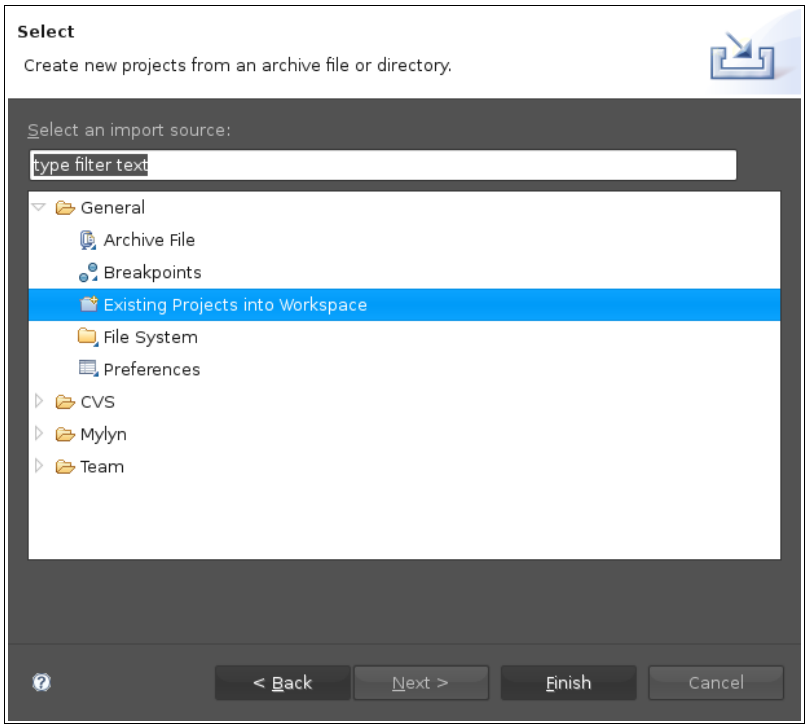
```
applicationCreator -eclipse ProyectoGWT client.MiAplicacion
```

Con esto crearás la base de tu aplicación, y un archivo de configuración para Eclipse, en donde se le indica al mismo el *cómo compilar tu proyecto*; luego deberás crear la estructura del proyecto en sí:

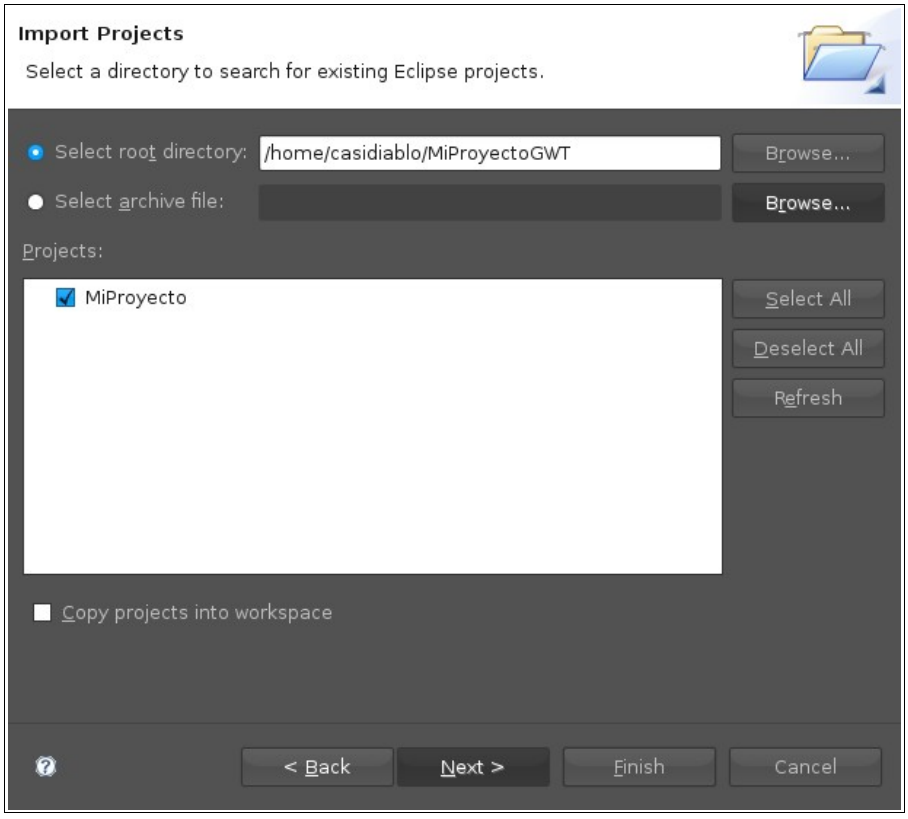
```
projectCreator -eclipse ProyectoGWT
```

El paso a seguir es añadir el proyecto a tu workspace del Eclipse. Para abrimos Eclipse, abrimos el menú File -> *Import*, luego seleccionamos "*Existing Projects*

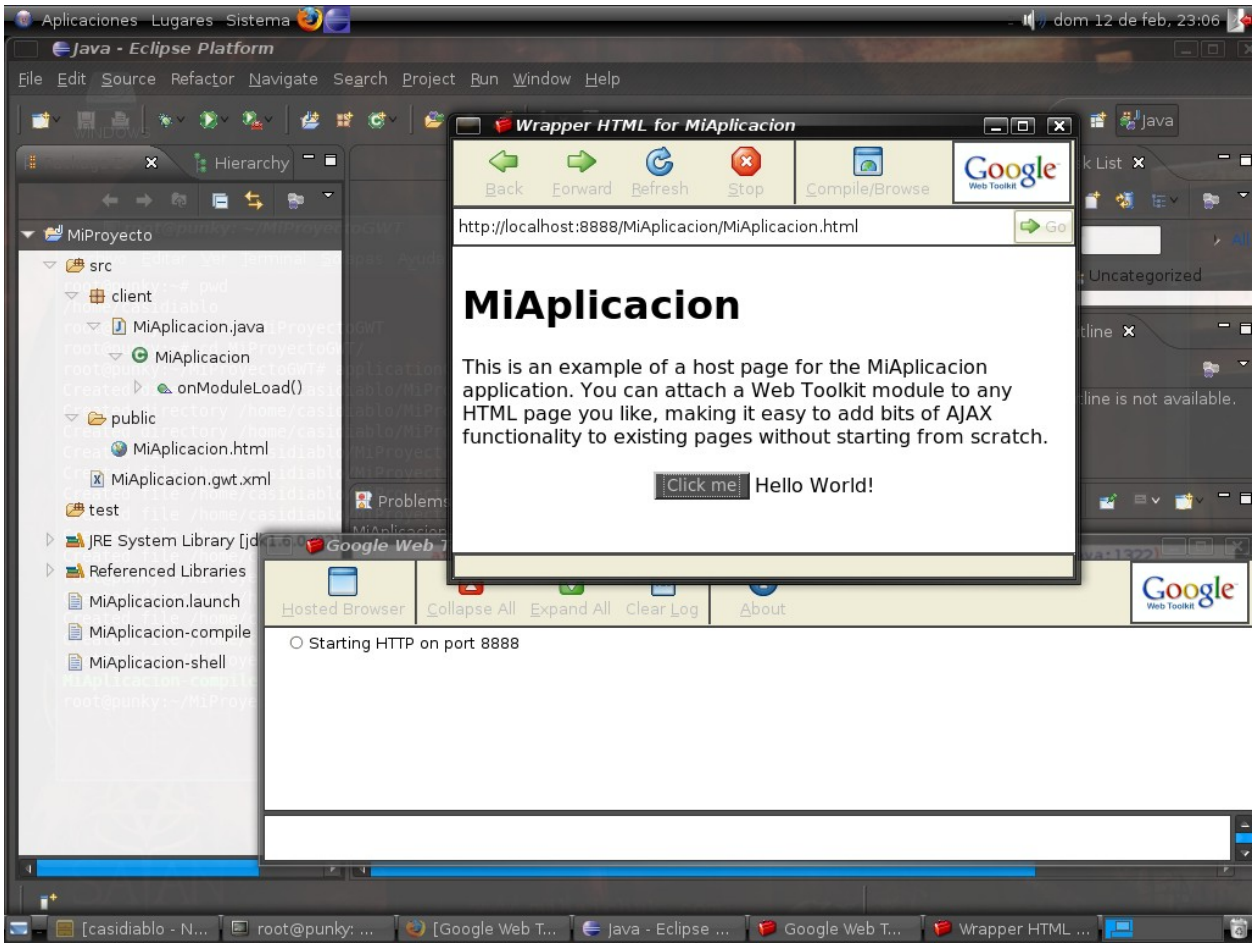
into Workspace":



Clic en *Next*, y luego seleccionamos la opción "*Select root directory*", y digitamos la ruta en donde hayamos creado el proyecto. Y si deseamos copiar los archivos de nuestra aplicación dentro del path definido como "workspace", pues marcas la opción "Copy projects into workspace", personalmente prefiero dejarlo allí:



Clic en en "*Finish*", y eso es todo! Ahora podrás crear, editar y depurar los archivos Java de tu aplicación, además de la posibilidad de ejecutar tu aplicación en modo hosted directamnete con el botón "Run":



Ahora solo queda que te pongas a desarrollar tu programa, y a disfutar de las ventajas que te ofrece un IDE como eclipse!

3. Guia para el desarrollador

En ésta entrada se lista la información importante acerca de muchas de las partes y/o conceptos que existen en Google Web Toolkit,

3.2 Lo fundamental de Google Web Toolkit

- El compilador de Google Web Toolkit
  Es un compilador que traduce tu código Java en código JavaScript
  - Soporte del lenguaje Java | Soporte al Runtime Library
- Soporte para DHTML
  La arcquitectura de Google Web Toolkit le permite a tus aplicaciones correr

de igual forma, sobre los distintos navegadores

- Depuración en modo hosted (hosted mode)
  Un navegador web incrustado en Google Web Toolkit, te permitirá correr y depurar tus aplicaciones en Java, antes de que estas sean convertidas a JavaScript
- Desarrollo en modo web (web mode)
  Convierte tu aplicación a JavaScript y pruébala sobre cualquier navegador
- Páginas HTML alojadas en un proyecto GWT
  Los módulos Google Web Toolkit son embebidos dentro de archivos HTML comunes
- Código del lado del cliente (client-side code)
  El código del lado del cliente, es el código ya traducido a JavaScript, listo para correr sobre un navegador
- Código del lado del servidor (server-side code)
  Se refiere al código que se ejecuta en el servidor en forma de ByteCodes
- Estructura de un proyecto GWT
  Se explica cómo debe ser la estructura interna de un proyecto de Google Web Toolkit
- Los módulos en GWT
  Los módulos son archivos XML que contienen las configuraciones de tu proyecto
  - Formato de los módulos XML | Inclusión automática de recursos | Filtrando paquetes públicos
- Herramientas de la línea de comandos
  Google Web toolkit proporciona una serie de herramientas que funcionan desde una consola (terminal), conoce cuales son y cómo se usan!
  - projectCreator | applicationCreator | junitCreator | i18nCreator | benchmarkCreator

c) El compilador de Google Web Tooltik

El corazón de Google Web Tooltik es un compilador que traduce el código Java en JavaScript, convirtiendo así tu aplicación desarrollada en Java en una aplicación equivalente en JavaScript.

En pocas palabras, si tu aplicación Google Web Tooltik compila y corre en "hosted mode" como tu esperabas, y si Google Web Tooltik compila/traduce tu aplicación a JavaScript sin problemas entonces tu aplicación está lista para correr sobre un navegador web desde cualquier servidor http.

El compilador Google Web Tooltik soporta la mayoría de las características del lenguaje de programación Java. Mientras que las librerías Google Web Tooltik runtime emulan una gran parte de las libreías de Java.

Las especificaciones

- *Soporte del Lenguaje:*
  Google Web Tooltik soporta la mayoría del núcleo del lenguaje Java 1.4 (semánticamente hablando), pero hay algunas diferencias de las que querrás darte cuenta.
- *Sorpote a librerías en tiempo de ejecución (Runtime Library):*

  Google Web Tooltik emula un conjunto básico de clases estándar de Java.

Soporte a Runtime Library

Google Web Toolkit soporta solamente una pequeña parte de las librerías de clases disponibles en Java 2 Standard Edition y Enterprise Edition, esto sucede ya que éstas librerías son bastantemente grandes y usan funcionalidades que no están disponibles en navegadores web. Para enterarse completamente de qué clases y métodos son soportados por el núcleo de paquetes Java, ver la referencia del API para java.lang y java.util, en donde se listan las clases a las que se les dio soporte y contiene notas acerca de las diferencias entre lo soportado y el estándar de Java.

Algunas áreas específicas en las que la emulación Google Web Toolkit difiere desde el estándar de Java son:

- **Expresiones regulares**
  La sintaxis para expresiones regulares en Java es similar, pero no idéntico, a las expresiones regulares en JavaScript. Por ejemplo, los métodos `replaceAll` y `split` usan expresiones regulares. Así que, te aconsejo que seas cuidadoso de usar expresiones regulares que tienen el mismo significado tanto en Java como en JavaScript.
- **Serialización**
  La serialización Java requiere de en unos cuantos mecanismos que no están disponibles en JavaScript, como por ejemplo el cargar dinámicamente clases y la reflexión. Por consiguiente, Google Web Toolkit no soporta el estándar de serialización de Java. En lugar de eso, Google Web Toolkit tiene una facilidad de soporte para RPC, que provee serialización automática de objetos desde y hacia el servidor usando invocación de métodos remotos.

Un consejo
Ahorrarás muchas frustraciones (y tiempo) si te aseguras de usar solamente las clases que son traducibles en el código del lado del cliente, desde el principio. Para ayudarte a identificar problemas anticipadamente, tu código es verificado contra la biblioteca de emulación JRE cada vez que corras tu aplicación en modo hosted. Como resultado, la mayoría de las veces que uses clases que no están soportadas serán notificadas la primera vez que intentes correr tu aplicación. Así que es mejor que pruebes el código a menudo, en modo hosted.

Soporte del Lenguaje Java

Google Web Toolkit compila el código de Java que es compatible con J2SE 1.4.2.

- **Tipos de datos nativos.**
  `byte`, `char`, `short`, `int`, `long`, `float`, `double`, `Object`, `String`, y arreglos son soportados. Sin embargo, no existen tipos integrales de 64-bit integral en JavaScript, así que las variables de tipo long son mapeadas a variables de coma flotante en JavaScript. Para asegurar la máxima consistencia entre el "modo hosted" y el "modo web", se recomienda que tu uses variables tipo `int`.
- **Excepciones.**
  `try`, `catch`, `finally` y excepciones definidas por el usuario son soportadas normalmente, aunque `Throwable.getStackTrace()` no es soportada por el modo web. Ver la sección de excepciones para detalles adicionales.
- **Sentencia assert.**
  El compilador Google Web Toolkit parséa las sentencias `assert`, pero no

emite código JavaScript para ellas.

- **Multi-hilado (subprocesamiento múltiple) y sincronización.**
  Los intérpretes de JavaScript usan single-threaded (uso de hilos simple), así que cuando Google Web Toolkit acepta la palabra `synchronized`, en realidad esto NO tiene ningún efecto sobre la aplicación final. Los métodos de las librerías de sincronización no están disponibles, incluyendo `Object.wait()`, `Object.notify()`, y `Object.notifyAll()`.
- **Reflexión.**
  Para un mejor desempeño, Google Web Toolkit compila tu código Java dentro de un script monolítico, y no soporta la carga subsecuente de clases. Ésta y otras optimizaciones excluyen el soporte general para la refexión. Es posible consultar el nombre de clase de un objeto usando el método Google Web `Toolkit.getTypeName(Object)`.
- **Finalización (recolección de basura).**
  JavaScript no soporta finalización durante la recolección de basura, así que Google Web Toolkit no puede usar nada similar a los finalizadores de Java (una de las principales características de este lenguaje) en modo web.
- **Strict Floating-Point.**

  La especificación del lenguaje de programación Java soporta variables de punto flotante (floating-point), incluyendo números de single-precision y double-precisión así como también la palabra reservada `strictfp`. Google Web Toolkit no tiene soporte para la palabra `strictfp` y no puede asegurar cualquier grado de precisión de punto flotante en el código traducido, así que lo mejor que puedes hacer es evitar en lo posible el uso de cálculos en el lado del cliente (en el navegador) que requieran un nivel alto de precisión.

d) Soporte a Cross-browser (DHTML)

Google Web Toolkit te protege de las preocupaciones que puedan surgir acerca de las incompatibilidades con cross-browser (DHTML compatible con todos los navegadores). Si estás construyendo widgets (por ejemplo), tus aplicaciones trabajarán de una manera similar en las recientes versiones de los navegadores Internet Explorer, Firefox, y Safari. (en Opera también, la mayoría de las veces). La construcción de interfaces de usuario DHTML no es para tomárselo a la ligera, por esto, es importante que pruebes tus aplicaciones sobre todos y cada uno de los navegadores antes de terminar de programar.

Siempre que sea posible, Google Web Toolkit utiliza los elementos nativos de la interfaz gráfica de usuario. Por ejemplo, el objeto botón de Google Web Toolkit es en realidad una etiqueta <> de HTML, en vez de un botón construido sintéticamente desde una etiqueta <>. Esto significa que los botones de Google Web Toolkit son renderizados apropiadamente en los distintos tipos de navegadores y sobre diferentes sistemas operativos. A los desarrolladores de Google Web Toolkit les gusta el uso nativo de controles ya que éstos son más rápidos, accesibles, y más familiares para los usuarios.

En lo que se refiere a dar estilo a aplicaciones Web, CSS es ideal. Así que, en vez de tratar de encapsular la estilización de la interfaz de usuario detrás de una pared de APIs, Google Web Toolkit provee muy pocos métodos directamente relacionados

con el estilo. En vez de ello, los desarrolladores están entusiasmados en definir hojas de estilos que son linkeadas al código de la aplicación usando nombres de estilo. Además de limpiar la aplicación, separando la presentación visual y el código, ésta división de labores ayuda a las aplicaciones a cargar y renderizarze más rápidamente, consume menos memoria, y hace más fácil el desarrollo impidiendo que nos tengamos que "devolver" a un punto, con el fin de cambiar el estilo.

Sección relacionada:

Hojas de estilo

e) Depuración en Modo Hosted (hosted mode)

Tú gastarás la mayor parte del tiempo de desarrollo en el **Modo Hosted**, que significa que estás interactuando y probando tu aplicación Google Web Toolkit sin que ésta haya sido traducida a JavaScript. En cualquier momento tú puedes editar, correr, y depurar aplicaciones desde un entorno de desarrollo integrado (IDE) de Java, mientras estás en "modo hosted". Al correr en modo hosted, la máquina virtual de (JVM) está actualmente ejecutando el código de tu aplicación como un archivo ByteCode compilado, usando Google Web Toolkit, que incrusta la aplicación en la ventana de un navegador. Si recordamos el típico ciclo de desarrollo de aplicaciones "codificar-probar-depurar", el modo hosted es de lejos la manera más productiva de desarrollar tu aplicación rápidamente.

> ***Un consejo***
> *En modo hosted, el desarrollo de aplicaciones Google Web Toolkit utiliza mucho la línea de comandos usando el classpath de la JVM's y tu path general. Asegúrate de tener la ruta donde está el núcleo de GWT y de tu aplicación en tu path cuando corras un proyecto.*

f) Desarrollo en Modo Web (web mode)

Mientras desarrollas una aplicación, debes "moverte" entre el desarrollo y el testeo de la misma, y empezarás a interactuar con tu aplicación en modo web muy a menudo. El modo Web se refiere a acceder a tu aplicación desde un navegador normal – donde se ejecuta solamente JavaScript – tal como se pretende que los usuarios lo hagan al final, cuando ya esté lista.

Para crear una versión de modo web de tu aplicación, debes compilarlo usando ya sea el botón "Compile/Browse" dentro de la ventana del modo hosted o desde el compilador de la línea de comandos `com.google.gwt.dev.GWTCompiler`.

El Modo Web demuestra como funcionaría tu aplicación después de terminada, es decir, cuando tu aplicación se ejecute sobre un navegador que solamente necesitará un intérprete de JavaScript y NO requerirá ningún plug-in o la JVM.

g) Páginas HTML alojadas en tu proyecto

Cualquier página HTML alojada en tu proyecto puede incluir código creado con Google Web Toolkit. Una página HTML normalmente se ve como esto:

```
<html>
<head>

<!-- Properties can be specified to influenvce deferred binding -->
<meta name='gwt:property' content='locale=en_UK'>

<!-- Stylesheets are optional, but useful -->
<link rel="stylesheet" href="Calendar.css">

<!-- Titles are optional, but useful -->
<title>Calendar App</title>

</head>
<body>

<!-- The fully-qualified module name, followed by 'nocache.js' -->
<script language="javascript"
src="com.example.cal.Calendar.nocache.js"></script>

<!-- Include a history iframe to enable full GWT history support -->
<!-- (the id must be exactly as shown) -->
<iframe src="javascript:''" id="__gwt_historyFrame"
style="width:0;height:0;border:0"></iframe>

</body>
</html>
```

La estructura fue diseñada para hacer que sea fácil de añadir las funcionalidades de Google Web Toolkit a aplicaciones web existentes con pocos cambios sobre las mismas.

h) Código del lado del cliente (Client-side code)

Una vez terminada, tu aplicación será enviada por la red a un usuario, donde correrá como JavaScript dentro del navegador de este. Todo lo que suceda dentro del navegador de tu usuario es llamado procesamiento del lado del cliente, o client-side processing.

Cuando escribes "client-side code" que está pensado para correr en el navegador web, recuerda que finalmente es convertido en código JavaScript. Por ende, es importante usar solamente librerías y lenguaje Java que puedan ser traducidos.

i) Código del lado del servidor (server-side Code)

Todo lo que suceda dentro del servidor es llamado procesamiento del lado del servidor, o server-side processing. Cuando tu aplicación tu aplicación necesita

interactuar con el servidor (por ejemplo, para cargar o guardar datos), ésta realiza una petición del lado del cliente (client-side request) desde el navegador, a través de la red usando invocaciones remotas a métodos (remote procedure call, RPC). Mientras se está procesando una llamada RPC, tu servidor está ejecutando código del lado del servidor.

> ***Un consejo***
> *Google Web Toolkit no se "entromete" en la posibilidad de correr códigos de bytes de Java (bytecodes) sobre el servidor. El código del lado del servidor (server-side code) no necesita ser traducido, por lo que eres libre de elegir cualquier librería para que consideres necesaria.*

j) Estructura de un proyecto

Los proyectos Google Web Toolkit están cubiertos con una capa de paquetes de Java. Si tú estás iniciando un proyecto Google Web Toolkit desde cero, debes usar la capa de paquetes estándar de Google Web Toolkit, que permite diferenciar fácilmente el código del lado del cliente del código del lado del servidor. Por ejemplo, supongamos que tu nuevo proyecto es llamado "Calendar". La capa de paquetes estándar deberá verse así:

Paquete	Propósito
`com/example/cal/`	El paquete raíz del proyecto contiene archivos del modulo en XML com/example/cal/client/ Código del lado del cliente y subpaquetes
`com/example/cal/server/`	Código del lado del servidor y subpaquetes
`com/example/cal/public/`	Recursos estáticos que pueden ser servidos públicamente

Y archivos de dentro de los paquetes así:

- **`com/example/cal/Calendar.gwt.xml`**
  Un módulo básico para tu proyecto que hereda del módulo com.google.gwt.user.User
- **`com/example/cal/CalendarApp.gwt.xml`**
  Hereda del modulo com.example.cal.Calendar (arriba) y añade una clase de punto de entrada (entry-point)
- **`com/example/cal/CalendarTest.gwt.xml`**
  Un módulo definido por tu proyecto
- **`com/example/cal/client/CalendarApp.java`**
  Código fuente Java del lado del servidor para la clase entry-point
- **`com/example/cal/client/spelling/SpellingService.java`**
  Un interfaz del servicio RPC service definida en un subpaquete
- **`com/example/cal/server/spelling/SpellingServiceImpl.java`**
  Código fuente Java que implementa la lógica del servicio de verificación de sintaxis
- **`com/example/cal/public/Calendar.html`**
  Una página HTML que carga la aplicación
- **`com/example/cal/public/Calendar.css`**
  Una hoja de estilo para la aplicación
- **`com/example/cal/public/images/logo.gif`**

Un logo

Un consejo
La mejor forma de crear un proyecto Google Web Toolkit desde cero es usar el "projectCreator script".

k) Módulos

Las unidades individuales (valga la redundancia) de configuraciones en GWT son archivos XML llamados módulos. Un módulo reúne todos los datos de configuración que tu proyecto GWT necesita, es decir:

- Módulos heredados
- Un nombre de clase; esto es opcional, aunque cualquier módulo referido a un HTML debe tener al menos una clase entry-point especificada.
- Entradas a los source paths
- Entradas a los public paths

Los módulos pueden aparecer en cualquier paquete en tu classpath, aunque es altamente recomendable que estos aparezcan en el paquete raíz de un proyecto estándar.

Clases entry-point

Un módulo **entry-point** es cualquier clase que es asignable a EntryPoint y que puede ser construida sin parámetros. Cuando un módulo es cargado, cada clase entry point es instanciada y el método EntryPoint.onModuleLoad() es llamado.

Source Path

Los módulos pueden especificar qué subpaquetes contienen código fuente traducible, provocando que el paquete nombrado y sus subpaquetes sean añadidos al source path. Solamente los archivos encontrados en el source path son candidatos para ser traducidos a JavaScript, haciendo posible que se mezclen códigos fuentes del lado del cliente (client-side) con los del lado del servidor (server-side) en el mismo classpath sin ningún tipo de conflicto.

Cuando un módulo hereda de otro, sus source path son combinados así que cada módulo tendrá acceso al código fuente traducible que requiera.

Public path

Los módulos pueden especificar qué subpaquetes son públicos, provocando que el paquete nombrado y sus subpaquetes sean añadidos al public path. Cuando compilas tú aplicación a JavaScipt, todos los archivos que pueden ser encontrados sobre tu public path son copiados al directorio de salida de los módulos. El efecto en la red es que las URLs visibles al usuario no necesitan incluir un nombre de paquete completo.

Cuando un módulo hereda de otro módulo, sus public paths son combinados así que cada módulo tendrá acceso al recurso estático que requiera.

Especificaciones

- Formato de módulos XML
  Los módulos son definidos en XML y situados dentro de la jerarquía de

paquetes de tu proyecto

- Inclusion Automática de paquetes
  Los módulos contienen referencias a archivos JavaScript y CSS externos, causando que estos sean cargados cuando el módulo mismo es cargado.
- Filtrado de paquetes públicos

  Filtra archivos dentro y fuera de tu public path para evitar la publicación accidental de archivos.

Formato de módulo XML

Los módulos son definidos en ficheros XML cuya extención de archivo es `.gwt.xml`. Los archivos de módulos XML deben residir en el paquete raiz de tú proyecto.

Si tu estás usando una estructura estándar de proyecto, tu módulo XML puede ser tan simple como esto:

```
<module>
<inherits name="com.google.gwt.user.User"/>
<entry-point class="com.example.cal.client.CalendarApp"/>
</module>
```

Cargando módulos

Los archivos de módulos XML son encontrados en el classpath de Java, referenciados por su propio nombre de módulo desde las páginas alojadas en el proyecto, y por ser heredados por otros módulos.

Los módulos son siempre referenciados por sus nombres lógicos. El nombre lógico de un módulo es del tipo `pkg1.pkg2.NombreModulo` (aunque pueden haber cualquier número de paquetes) y excluyendo incluso la extensión del archivo. Por ejemplo, el nombre lógico de un archivo de módulo XML situado en

`~/src/com/ejemplo/cal/Calendario.gwt.xml`

es

`com.ejemplo.cal.Calendario`

Elementos disponibles

- `<inherits name="logical-module-name"/>`
  Hereda todas las configuraciones desde el módulo especificado como si el contenido del módulo XML heredado fuera copiado literalmente. Cualquier número de módulos pueden ser heredados de ésta forma.
- `<entry-point class="classname"/>`
  Especifica una clase entry point. Cualquier número de clases entry-point pueden ser añadidas, incluyendo los provenientes de módulos heredados.
- `<source path="path"/>`
  Añade paquetes al source path para combinar los paquetes en que el módulo XML es encontrado con el path especificado al subpaquete. Cualquier archivo de Java que aparezca en este subpaquete o cualquiera de sus subpaquetes son traducidos.

Si no se define ningún elemento <source> en un archivo de módulo XML, el subpaquete `client` es implicitamente añadido al source path como si <source path="client"> huviera sido encontrado en el XML.

- `<public path="path"/>`
  Añade paquetes al public path para combinar el paquete en el que el módulo XML es encontrado con el path especificado para identificar la raiz del public path. Cualquier archivo que aparezca en este paquete o cualquiera de sus subpaquetes será tratado como un recurso de acceso público. El elemento `<public>` soporta filtrado pattern-based para permitir control de fine-grained sobre los recursos copiados al directorio de salida durante el proceso de compilación.
  Si no se define un elemento `<public>` en un archivo de módulo XML, el subpaquete public es añadido implicitamente al public path como si `<public path="public">` hubiera sido encontrado en el XML.
- `<servlet path="url-path" class="classname"/>`
  Para el uso conveniente de RPC, este elemento carga una clase servlet montada como el path URL especificado. El path URL debe ser absoluto y tener la forma de un directorio (por ejemplo, /spellchek). Tu código cliente entonces, especifica éste mapeo de URL en una llamada al método `ServiceDefTarget.setServiceEntryPoint(String)`. Cualquier número de servlets puede ser cargado de ésta manera, incluyedo los que módulos heredados.
- `<script src="js-url"/>`
  Injecta automaticamente archivos JavaScript externos localizados en la ruta src. Ver Inclusión automatica de recursos para más detalles.
- `<stylesheet src="css-url"/>`
  Injecta automaticamente hojas de estilo en cascada situadas en archivos externos, localizados en la ruta src. Ver Inclusión automatica de recursos para más detalles.
- `<extend-property name="client-property-name" values="comma-eparated-values"/>`
  Extiende el conjunto de valores para una propiedad del cliente existente. Cualquier número de valores puede ser añadido de ésta manera, y los valores de la propiedad del cliente acmumulado desde módulos heredados. Tú querrás usar esto si vas a crear un proyecto con Internazionalización.

Inclusión de recursos automático

Los módulos pueden contener referencias a archivos JavaScript y CSS externos, causando que estos sean cargados automáticamente cuando el módulo mismo es cargado.

Incluyendo JavaScript Externo

La inclusión de scripts es una forma conveniente de asociar archivos JavaScript externos con tu modulo de manera automática. Usa la siguiente sintaxis para cargar un archivo JavaScript externo dentro de la página alojada en tú proyecto, antes de que el entry point de tu módulo sea llamado:

```
<script src="js-url"/>
```

El script es cargado dentro del namespace de la página como si lo hubieras incluido

explícitamente usando la etiqueta `<script>` de HTML. El script será cargado antes que el método onLoadModule() sea llamado.

Incluyendo Hojas de Estilo Externas

La inclusión de hojas de estilo es una excelente forma de asociar archivos CSS externos con tú módulo de una manera automática. Usa la siguiente sintaxis para añadir archivos CSS externos a la página de tú módulo:

```
<stylesheet src="css-url"/>
```

Puedes añadir cualquier número de hojas de estilo de ésta manera, y el orden de la inclusión dentro de la página refleja el orden en que estos elementos aparecen en tú módulo XML.

Inclusión y herencia de módulos

La herencia de módulos crea recursos de inclusión particularmente convenientes. Si deseas crear una librería re-utilizable que depende de archivos JavaScript y CSS particulares, puedes estar seguro que los clientes de tú librería tienen todo lo que necesitan automáticamente, usando herencia desde tú módulo.

> ***Un consejo***
> *Versiones de GWT posteriores a la versión 1.4 requieren una función script-ready para determinar cuando un script incluido fue cargado. Ahora esto ya no es necesario; todos los scripts incluidos serán cargados cuando la aplicación inicie, en el orden en el que estos fueron declarados.*

Entradas relacionadas

- Formato del módulo XML

Filtrando paquetes públicos (public packages)

Los elementos `<public>` soportan ciertos atributos y elementos anidados para permitir patrones basados en inclusión y exclusión. Ésto utiliza las mismas reglas que un elemento `FileSet` de Ant. Por favor leer la documentación de FileSet para una visión general.

El elemento `<public>` no soporta completamente la semántica de `FileSet`. Solamente los siguientes atributos y elementos anidados son soportados:

- El atributo `include`
- El atributo `exclude`
- El atributo `defaultexcludes`
- El atributo `casesensitive`
- Etiquetas `include` anidadas
- Etiquetas `exclude` anidadas

Los demás atributos y elementos anidados no son soportados.

Importante
El valor predeterminado de `defaultexcludes` es `true`. Por defecto, los patrones listados aquí son excluidos.

I) Herramientas de la línea de comandos

GWT te ofrece un pequeño set de herramientas de línea de comandos fáciles de manejar, para realizar diferentes tareas de una manera rápida. Éstas son también útiles para añadir nuevas cosas a los proyectos existentes. Por ejemplo, `projectCreator` podría ser usado para crear un proyecto Eclipse para uno de los ejemplos que viene con GWT.

projectCreator
Genera el esqueleto de un proyecto básico y un archivo Ant opcional y/o proyecto para Eclipse.

aplicationCreator
Genera el lanzador de una aplicación

junitCreator
Genera un test Junit

i18nCreator
Genera un archivo de propiedades i18n y un script de sincronización

benchmarkViewer
Muestra resultados benchmark

projectCreator

Genera el esqueleto de un proyecto básico y, un archivo Ant opcional y/o proyecto para Eclipse.

```
projectCreator [-ant projectName] [-eclipse projectName] [-out dir]
[-overwrite] [-ignore]


-ant Genera un archive Ant para compilar las fuentes (su extensión es
.ant.xml)
-eclipse Genera un proyecto Eclipse
-out El directorio donde se generarán los archivos de salida (por defecto
la ruta actual)
-overwrite Sobre escribe los archivos existentes
-ignore Ignora cualquier archivo existente; no lo sobrescribe
```

Ejemplo

```
~/miproyecto> projectCreator -ant MiProyecto -eclipse MiProyecto
Created directory src
Created directory test
Created file Foo.ant.xml
Created file .project
Created file .classpath
```

Ejecutando `ant -f MiProyecto.ant.xml` se compilará `src` dentro de `bin`. Usando el archivo ant, y con las opciones adecuadas, es posible compilar un proyecto a un archivo .jar.

El archivo `.project` puede ser importado dentro de un workspace de Eclipse.

applicationCreator

Genera un lanzador de aplicación y scripts para ejecutar el modo hosted y compilar a JavaScript.

```
applicationCreator [-eclipse projectName] [-out dir] [-overwrite]
[-ignore] classname

-eclipse Crea un lanzador de depuración para el proyecto Eclipse nombrado
-out El directorio donde se escribirán los archivos de salida
-overwrite Sobrescribe archivos existentes
-ignore Ignora cualquier archive existente; no sobrescribe
className El nombre completo de la clase a crear
```

Ejemplo

```
~/MiProyecto> applicationCreator -eclipse MiProyecto
com.example.foo.client.MiProyecto
Created directory src/com/example/miproyecto/client
Created directory src/com/example/miproyecto/public
Created file src/com/example/miproyecto/MiProyecto.gwt.xml
Created file src/com/example/miproyecto/public/MiProyecto.html
Created file src/com/example/miproyecto/client/MiProyecto.java
Created file MiProyecto.launch
Created file MiProyecto-shell
Created file MiProyecto-compile
```

Ejecutando `MiProyecto-shell` sube la nueva aplicación en modo hosted. `MiProyecto-compile` traduce la aplicación Java a JavaScript, creando una carpeta en www. `MiProyecto.launch` es un lanzador para Eclipse.

junitCreator

Genera un test JUnit y scripts para probar la aplicación en modo hosted y en modo web.

```
junitCreator -junit pathToJUnitJar [-eclipse projectName] [-out dir]
[-overwrite] [-ignore] className

-junit Especifica el path a tu junit.jar (requerido)
-module Especifica el nombre del módulo de la aplicación a usar
(requerido)
-eclipse Crea un archivo de depuración para el proyecto eclipse citado
-out El directorio donde se escribirán los archivos de salida
-overwrite Sobrescribe archivos existentes
-ignore Ignora cualquier archive existente; no sobrescribe
className El nombre completo de la clase a crear
```

Ejemplo

```
~/MiProyecto> junitCreator -juni
/opt/eclipse/plugins/org.junit_3.8.1/junit.jar -module
com.example.miproyecto.MiProyecto -eclipse MiProyecto
com.example.miproyecto.client.MiProyectoTest
Created directory test/com/example/miproyecto/test
Created file test/com/example/miproyecto/client/MiProyectoTest.java
Created file MiProyectoTest-hosted.launch
Created file MiProyectoTest-web.launch
Created file MiProyectoTest-hosted
Created file MiProyectoTest-web
```

Ejecutando `MiProyectoTest-hosted` se prueban los bytecodes de Java en una Máquina Virtual (JVM). `MiProyectoTest-web` verifica el JavaScript compilado.

i18nCreator

Genera scripts de internacionalización para internacionalización estática, usando archivos de propiedades de Java como ejemplo.

```
i18nCreator [-eclipse projectName] [-out dir] [-overwrite] [-ignore]
[-createMessages] interfaceName

-eclipse Crea un archivo de depuración para el proyecto eclipse citado
-out El directorio donde se escribirán los archivos de salida
-overwrite Sobrescribe archivos existentes
-ignore Ignora cualquier archive existente; no sobrescribe
className El nombre completo de la clase a crear
-createMessages Genera scripts para una interfaz de "Messages" en vez de
un "Constants"
interfaceName El nombre completo de la interfaz creada
```

Ejemplo

```
~/MiProyecto> i18nCreator -eclipse MiProyecto -createMessages
com.example.miproyecto.client.MiProyectoMessages

Created file
src/com/example/miproyecto/client/MiProyectoMessages.properties
Created file MiProyectoMessages-i18n.launch
Created file MiProyectoMessages-i18n

~/MiProyecto> i18nCreator -eclipse MiProyecto
com.example.miproyecto.client.MiProyectoConstants
Created file
src/com/example/miproyecto/client/MiProyectoConstants.properties
Created file MiProyectoConstants-i18n.launch
Created file MiProyectoConstants-i18n
```

benchmarkViewer

Lee un reporte benchmark desde una carpeta y muestra sus resultados, incluyendo varias visualizaciones.

```
benchmarkViewer [path]


path Especifica la ruta del archivo XML de reportes benchmark. Si no se
especifica un path intentará buscar en la ruta actual.
```

Ejemplo

```
~/MiProyecto> benchmarkViewer mi/benchmark/resultados
```

Con esto se mostrarán los resultados que se encuentre en el archive de reportes de la carpeta mi/benchmark/resultados.

3.3 Construyendo Interfaces de Usuario

Las clases de interfaces de usuario de Google Web Toolkit son muy similares a las de los distintos frameworks como Swing o SWT, excepto que los widgets son creados y renderizados usando HTML creado dinámicamente.

Mientras sea posible manipular el DOM del navegador directamente usando la interfaz DOM, es más fácil usar las clases desde la jerarquía de widgets. Raramente necesitarás usar DOM directamente. Usando widgets puedes construir interfaces de usuario rápidamente, y que se comporten de manera adecuada sobre todos lo navegadores.

Especificaciones

- Widgets y paneles
  Los widgets y paneles son códigos en Java del lado del cliente usuados para construir interfaces de usuario.
- Galería de widgets
  En esta entrada se listan los objetos de clase para construir interfaces de usuario
- Events y Listeners
  Los widgets pueden emitir eventos (Events) que son escuchados (Listener) por los manejadores de eventos respectivos.
- Creando Widgets personalizados
  Crea tu propio widget completamente usando Java
- Entendiendo los layouts
  En esta entrada se exponen los contenedires de Widgets más utilizados
- Hojas de estilo
  Del cómo se usan las Hojas de Estilo en Cascada en Google Web Toolkit
- Atado de imágenes (image bundle)

  Optimiza el funcionamiento de tu aplicación reduciendo el número de peticiones HTTP al servidor

c) Widgets y paneles

En las aplicaciones de GWT se construyen interfaces de usuario usando widgets dentro de paneles. Se denominan widgets a objetos como Button (botones),

TextBox (cajas de texto) y Tree (árboles).

Los widgets y paneles trabajan de igual manera sobre los diferentes navegadores, usándolos, eliminas la necesidad de escribir código especial para cada navegador. Aún así, no estás limitado al conjunto de widgets que vienen incluidos en este toolkit, existen varias formas de crear un widget personalizado.

Paneles

Los paneles como DockPanel, HorizontalPanel y RootPanel, están diseñados para contener widgets dentro y son usados para determinar como la interfaz de usuario va a ser distribuida sobre el navegador.

Estilos

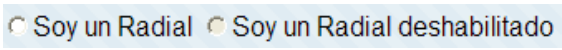
Estilos visuales pueden ser aplicados a las interfaces de usuario usando Hojas de Estilo en Cascada (CSS). En ésta sección se describe como usar esta característica.

d) Galería de Widgets

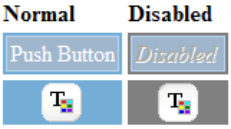
Los siguientes son Widgets y Paneles disponibles en la librería de clases de GWT:

Button

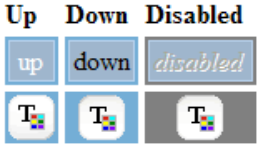
RadioButton



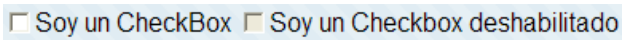
PushButton



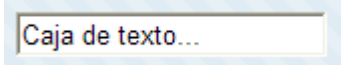
ToggleButton



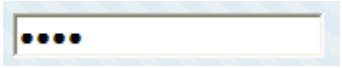
CheckBox



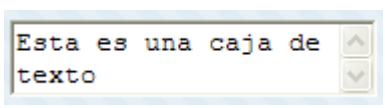
TextBox



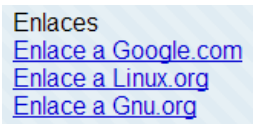
PasswordTextBox



TextArea

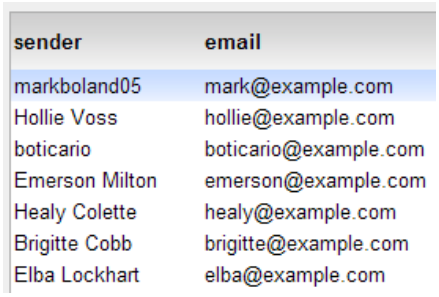


Hyperlink



MenuBar

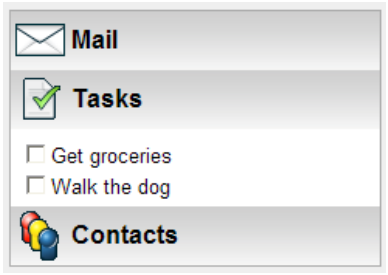
Table



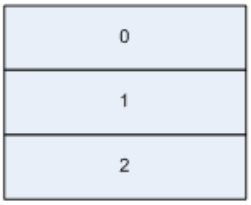
sender	email
markboland05	mark@example.com
Hollie Voss	hollie@example.com
boticario	boticario@example.com
Emerson Milton	emerson@example.com
Healy Colette	healy@example.com
Brigitte Cobb	brigitte@example.com
Elba Lockhart	elba@example.com

DialogBox

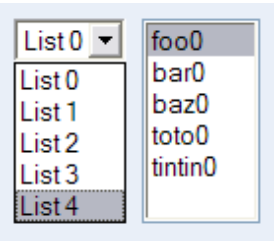
StackPanel



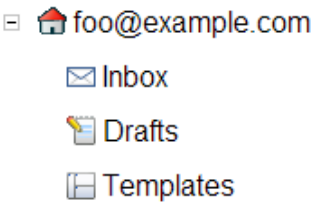
VerticalPanel



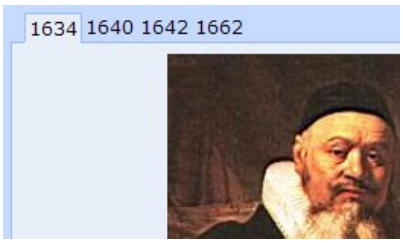
ListBox



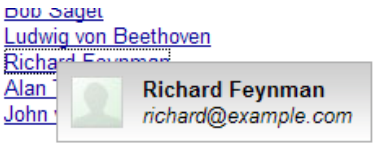
Tree



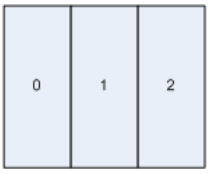
TabBar



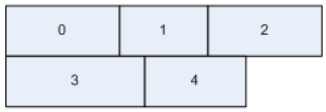
PopupPanel



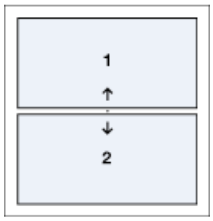
HorizontalPanel



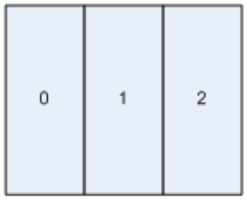
FlowPanel



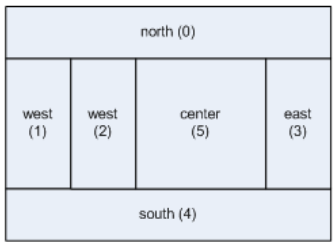
VerticalSplitPanel



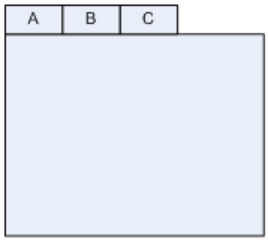
HorizontalSplitPanel



DockPanel

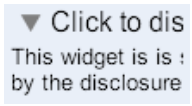


TabPanel

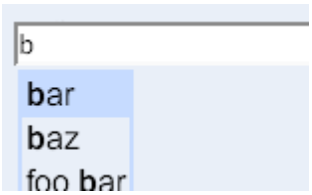


RichTextArea

DisclosurePanel



SuggestBox



La mayoría de las imágenes aquí expuestas fueron tomadas de ésta web. Si deseas ver una lista más extensa con diversos Widgets creados por el staff del proyecto Google Web Toolkit y otros creados por usuarios, puedes visitar la siguiente web:

http://www.gwtpowered.org/#Resources

e) Events y Listeners

Los eventos en Google Web Toolkit usan interfaces listener (de escucha de eventos), de una manera muy similar a como se utilizan en otros frameworks. Una interfaz listener, define uno o más métodos que los widgets usan para anunciar un evento. Una clase que recibe eventos de un tipo en particular, implementa la interfaz listener asociada con el tipo de eventos que recibe y luego pasa una referencia del widget que generó el evento para "suscribirlo" a un conjunto de eventos.

La clase Button, por ejemplo, publica eventos click. La interfaz asociada para este tipo de eventos es ClickListener:

```
public void ejemploDeClickListener()

{
  Button b = new Button("Dame clic");
  b.addClickListener(new ClickListener() {
    public void onClick(Widget sender) {
      // aquí se determina que sucede cuando se hace clic
    }
  });
}
```

Usar clases anónimas dentro de otras clases, puede ser ineficiente para un gran número de widgets, puesto que podría resultar en la creación de muchos objetos listener. Los widgets proveen su apuntador `this` como el parámetro `sender` cuando invocan un método listener, permitiendo que sea sencillo para el listener distinguir entre los diferentes publicadores. Esto hace un mejor uso de la memoria, pero requiere más lineas de código para su implementación, como en el siguiente ejemplo:

```
public class EjemploListener extends Composite implements ClickListener {
private FlowPanel fp = new FlowPanel();
private Button b1 = new Button("Button 1");
private Button b2 = new Button("Button 2");

public EjemploListener() {
initWidget(fp);
fp.add(b1);
fp.add(b2);
b1.addClickListener(this);
b2.addClickListener(this);
}

public void onClick(Widget sender) {
if (sender == b1) {
// handle b1 being clicked
} else if (sender == b2) {
// handle b2 being clicked
}
}
}
```

Algunas interfaces de eventos especifican más de un evento. Si estás interesado en solo un subconjunto de eventos, puedes usar “adapters”. Los adaptadores (adapters) son implementaciones vacías de una interfaz de eventos en particular, desde que puedas derivar una clase listener sin tener que implementar todo el método:

```
public void ejemploAdapter() {
TextBox t = new TextBox();
t.addKeyboardListener(new KeyboardListenerAdapter() {
public void onKeyPress(Widget sender, char keyCode, int modifiers) {
// handle only this one event
}
});
}
```

f) Creando Widgets personalizados

GWT permite que la creación de widgets personalizados sea muy fácil, usando el lenguaje Java.

Composites

Los composites son de lejos la manera más efectiva de crear widgets. Puedes fácilmente combinar grupos de widgets dentro de un composite que es en sí mismo un widget re-utilizable. Composite, es un widget especial que puede contener otros componentes (por lo general, un panel) pero se comporta como si el fuera su widget contenido. Es preferible usar Composite que intentar crear widgets complejos usando subclases de panel ya que un Composite usualmente controla de una mejor forma, qué métodos son publicados.

Desde Cero en código Java

También es posible crear un widget desde cero usando código en Java, aunque este método es más complicado, ya que tienes que escribirlo a bajo nivel. Muchos de los widgets básicos son escritos de ésta manera, como Button o TextBox. De momento no hay mucha documentación acerca de cómo crear widget de esta forma, pero siempre puedes referirte a la implementación de la clase Button (por ejemplo), para entender su funcionamiento.

Usando JavaScript

Cuando se implementa un widget personalizado que deriva directamente desde la clase Widget, puedes escribir algunos widgets usando JavaScript. Esto debe ser utilizado únicamente como un último recurso, además es conviene considerar las implicaciones que puedan tener los métodos que escribes sobre el tipo de navegador usado, es decir, ser cuidadoso para que tu código corra sobre cualquier navegador; y no solo eso, crear widgets con JavaScript también es más difícil de depurar. Para un ejemplo de esto, ver el widget TextBox y su implementación baja.

g) Entendiendo los layouts

Los paneles en GWT son similares a sus contra parte en otras clases de librerías. La principal diferencia reside en el hecho de que estos usan elementos HTML como `DIV` o `TABLE` para disponer los widgets.

RootPanel

El primer panel que probablemente usarás es RootPanel. Éste panel además es la raíz de la jerarquía de paneles. El `RootPanel` por defecto envuelve el cuerpo del documento HTML, y es obtenido invocando el método RootPanel.get(). Si necesitas envolver otro elemento del documento HTML con `RootPanel`, tu puedes usar el método RootPanel.get(String).

CellPanel

CellPanel es la clase abstracta de DockPanel, HorizontalPanel, y VerticalPanel. Lo que tienen estos paneles en común es que posicionan sus widgets dentro de celdas lógicas. Por ende, un widget hijo puede ser puede ser alineado dentro de la celda que lo contiene, usando el método setCellHorizontalAlignment(), y setCellVerticalAlignment(). Los CellPanel también te permiten configurar el tamaño de las celdas usando CellPanel.setCellWidth y CellPanel.setCellHeight.

Otros paneles

Otros paneles incluyen DeckPanel, TabPanel, FlowPanel, HTMLPanel, y StackPanel.

Tamaños y medidas

Es posible asignar un tamaño explicitamente a un widget usando los métodos setWidth(), setHeight(), y setSize(). Los argumentos para esos métodos son strings, en vez de enteros, ya que así aceptan cualquier valor CSS válido, como valores de píxeles (214px), centímetros (3cm), y porcentajes (80%).

h) Hojas de estilo

Los widgets de GWT confian su estilo visual a hojas de estilo en cascada. Cada widget tiene un nombre de estilo asociado que lo enlaza a una regla CSS. Un nombre de estilo de widget es asignado usando el método setStyleName(). Por ejemplo, Button tiene como nombre de estilo por defecto `gwt-Button`. En ese orden de ideas, si tu desearas por ejemplo asignar un tamaño específico a la *letra* (fuente) de todos los botones de tu aplicación, podrías usar la siguiente regla en tu archivo CSS:

```
.gwt-Button { font-size: 150%; }
```

Estilos complejos

Algunos widgets tienen asociados tipos de estilos más complejos. MenuBar, por ejemplo, tiene los siguientes estilos:

```
.gwt-MenuBar { the menu bar itself }
.gwt-MenuBar .gwt-MenuItem { menu items }
.gwt-MenuBar .gwt-MenuItem-selected { selected menu items }
```

En este ejemplo, se tienen dos reglas que aplican a los items de los menús. El primero aplica a todos los items de los menús, mientras el segundo (con el sufijo `-selected`) aplica solo para el item de menú que esté seleccionado. Un nombre de estilo de ítem de menú seleccionado será asignado a "`gwt-MenuItem gwt-MenuItem-selected`", especificando que ambas reglas de estilo serán aplicadas. La manera más común de hacer esto es usar setStyleName para asignar el estilo de nombre base, luego addStyleName() y removeStyleName() para añadir y remover el segundo nombre de estilo.

Archivos CSS

Generalmente, las hojas de estilo son situados en paquetes que son parte del módulo *public path*. Entonces simplemente incluye una referencia a la hoja de estilo en tu página HTML, así:

```
<link rel="stylesheet" href="mystyles.css" type="text/css">
```

i) Atado de imágenes (image bundle)

Generalmente, una aplicación utiliza muchas imágenes pequeñas para iconos. Una petición al servidor debe ser enviada por cada una de las imágenes, y en algunos casos, el tamaño de la imagen es menor que la cabecera de la petición en sí, que es enviada con los datos de la imagen.

Éste tipo de peticiones son muy ineficientes para el servidor. Además, Siempre que las imágenes hayan sido guardadas en caché por el cliente, una petición 304 ("no modificado") es enviada para verificar que la imagen no haya sido modificada. Puesto que las imágenes no cambian muy a menudo, estas peticiones de verificaciones son también malas para el servidor.

Enviar muchas peticiones al servidor, para recibir imágenes y comprobar que no han cambiado hace que tu aplicación sea muy lenta. HTTP 1.1 hace que los navegadores limiten el número de conexiones HTTP a dos por dominio/puerto. Una gran cantidad de peticiones de imágenes satura las conexiones disponibles del navegador, lo cual bloquea las peticiones RPC de la aplicación. Las peticiones RPC son un trabajo más importante que tu aplicación debe realizar.

Para resolver este problema, GWT introduce el concepto de imágenes atadas (image bundle). Una imagen atada, es un conjunto de imágenes dentro de una sola imagen, con una interfaz para accesar a las imágenes individualmente. De esta forma, en vez de hacer una petición al servidor por cada imagen que necesites, la aplicación realiza solo una petición y recupera todas las imágenes.

Ya que los nombres de archivo de la composición de imágenes está basado en un *hash* del contenido del archivo, el nombre de archivo cambiará solamente si la imagen es cambiada. Esto significa que es seguro para los clientes guardar en caché la composición de imágenes permanentemente, lo que evita estar verificando innecesariamente si las imágenes han sufrido cambios en el servidor. Para lograr esto, la configuración del servidor necesita especificar que la composición de imágenes nunca expira. Además de la velocidad de carga de tu aplicación, usar image bundle previene el efecto "bouncy" al cargar las imágenes en el navegador.

Especificaciones

- Creando y usando image bundle
  Define un *image bundle* y úsalo en tu aplicación
- Image bundle y localización

Crea *locale-sensitive image bundle* para usar las capacidades de localización de GWT

Entradas relacionadas

ImageBundle

Creando y usando image bundle

Para definir un *image bundle*, el usuario necesita extender la interfaz ImageBundle. La interfaz ImageBundle es una etiqueta que puede ser extendida para definir nuevas ImageBundle.

La interfaz derivada puede tener cero o más métodos, donde cada método:

- no toma parámetros
- tiene un tipo de retorno AbstractImagePrototype, y
- puede tener un metadata `gwt.resource` opcional lo que especifica el nombre del archivo de imagen en el classpath del módulo.

Los tipos de archivo válidos son `png`, `gif`, y `jpg`. Si el nombre de la imagen contiene caracteres "/", se asume que es un recurso en el classpath, formateado como lo hubiera hecho `ClassLoader.getResource(String)`. De otra forma, la imagen debe estar situada en el mismo paquete que el usuario ha definido.

Si la metadata `gwt.resource` no es especificada, entonces

- se supone que el nombre de archivo de la imagen es comparada con el nombre del método
- se asume que la extensión es `.png`, `.gif` o `.jpg` y
- que el archivo está en el mismo paquete que la interfaz derivada

En caso de que allá múltiples archivos de imagen con diferentes extensiones, el orden de precedencia de extensiones es (1) `png`, (2) `gif` y (3) `jpg`.

Un ImageBundle para los iconos de un procesador de texto podría ser definido así:

```
public interface WordProcessorImageBundle extends ImageBundle {

/**
* Would match the file 'new_file_icon.png', 'new_file_icon.gif', or
* 'new_file_icon.png' located in the same package as this type.
*/
public AbstractImagePrototype new_file_icon();

/**
* Would match the file 'open_file_icon.gif' located in the same
* package as this type.
*
* @gwt.resource open_file_icon.gif
```

```
*/
public AbstractImagePrototype openFileIcon();

/**
* Would match the file 'savefile.gif' located in the package
* 'com.mycompany.mygwtapp.icons', provided that this package is part
* of the module's classpath.
*
* @gwt.resource com/mycompany/mygwtapp/icons/savefile.gif
*/
public AbstractImagePrototype saveFileIcon();
}
```

Los métodos en un *image bundle* retornan objetos `AbstractImagePrototype` (en vez de objetos `Image`, como se podría esperar), ya que los objetos `AbstractImagePrototype` proveen un aligeramiento adicional del peso de la imagen. Por ejemplo, el método AbstractImagePrototype.getHTML() provee un fragmento HTML representa una imágen sin tener que crear una instancia del actual widget de imagen. En algunos casos, esto puede ser más eficiente para administrar imágenes usando esos fragmentos de HTML.

Otro uso para `AbstractImagePrototype` es utilizar `AbstractImagePrototype.applyTo(Image)` para transformar un objeto `Image` exitente dentro de uno que encaje con el prototipo sin tener que instanciar otro objeto `Image`. Esto puede ser útil si tu aplicación posee imágenes que necesiten ser cambiadas de acuerdo a alguna acción por parte del usuario. De momento, si lo que necesitas es un objeto `Image`, el método `AbstractImagePrototype.createImage()` puede ser usado para generar nuevas instancias de `Image`.

El siguiente ejemplo muestra como usar el image bundle que definimos en tu aplicación:

```
public void useImageBundle() {
WordProcessorImageBundle wpImageBundle = (WordProcessorImageBundle)
GWT.create(WordProcessorImageBundle.class);
HorizontalPanel tbPanel = new HorizontalPanel();
tbPanel.add(wpImageBundle.new_file_icon().createImage());
tbPanel.add(wpImageBundle.openFileIcon().createImage());
tbPanel.add(wpImageBundle.saveFileIcon().createImage());
}
```

Related topics

- ImageBundle
- AbstractImagePrototype

3.4 RPC: Invocación de procedimientos remotos

El concepto de RPC es muy similar al RMI. Una diferencia fundamental entre Google Web Toolkit y las tradicionales aplicaciones web, es que las aplicaciones Google Web Toolkit no necesitan de otras páginas web mientras son ejecutadas. Ya que las páginas construidas con Google Web Toolkit corren como aplicaciones sobre el navegador, éstas no necesitan hacer nuevas peticiones al servidor para, por

ejemplo, realizar actualizaciones en la interfaz de usuario. Sin embargo, como todas las aplicaciones cliente/servidor, los programas en Google Web Toolkit necesitarán pedir ciertos datos al servidor para realizar determinadas tareas. El mecanismo para interactuar con el servidor a través de la red es llamado "remote procedure call" (RPC), que viene siendo en español "llamada a procedimiento remoto". El RPC en Google Web Toolkit permite fácilmente al cliente enviar y recibir objetos de Java sobre HTTP.

Cuando es usado adecuadamente, RPC te da la oportunidad de mover toda la lógica de la interfaz de usuario al cliente, lo que mejora el funcionamiento de la aplicación, reduce el ancho de banda usado, reduce la carga al servidor, y le presenta al usuario final una experiencia más agradable navegando por la página.

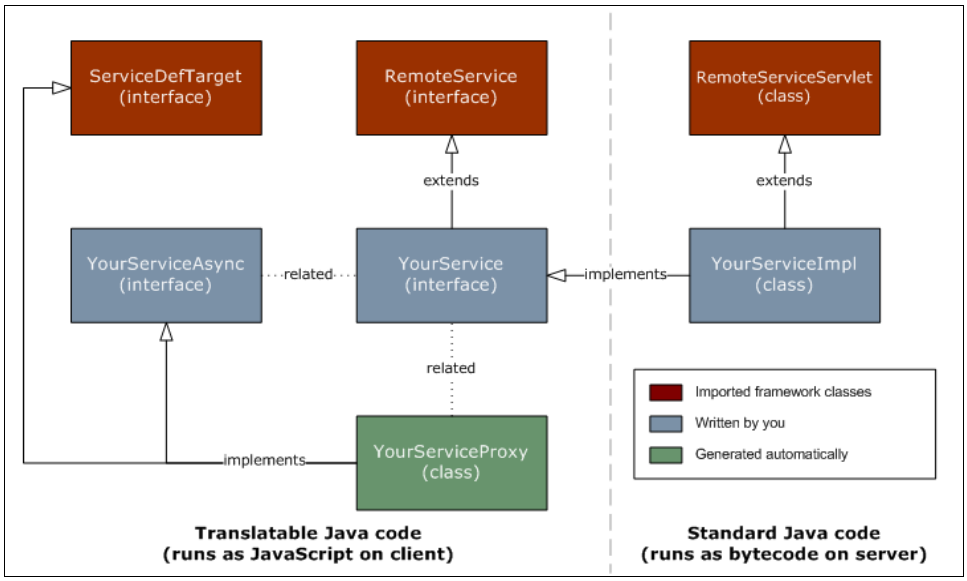
El código del lado del servidor que es invocado desde el cliente es frecuentemente llamado "servicio", por lo que el "llamar procedimientos remotos" es comúnmente llamado como invocación de servicios. Aunque es necesario tener claro que el término "servicio" en este contexto NO tiene el mismo concepto de "web-service". En realidad, los servicios Google Web Toolkit no son lo mismo que "Simple Object Access Protocol" (SOAP).

Aquí tienes más información acerca del uso de RPC:

- La anatomía de RPC en Google Web Toolkit
- Creación de servicios
- Implementación de servicios
- Realizando invocaciones a servicios

c) Anatomía de RPC

En ésta sección se esbozan las partes básicas requeridas para invocar un servicio. Cada servicio tiene una pequeña familia de interfaces y clases. Algunas de esas clases, como el servicio Proxy, son generados automáticamente y generalmente nunca te percatarás de que existen. El patrón de clases es idéntico para todos los servicios que implementas, por lo que es una buena idea familiarizarse un poco con los términos y el propósito de cada capa, en el procesamiento de llamadas al servidor. Si estás familiarizado con el mecanismo tradicional de RPC, probablemente ya conocerás la mayoría de la terminología aquí expuesta. (*El gráfico fue tomado de la página oficial de Google Web Toolkit.*)



d) Creando servicios

Para desarrollar una nueva interfaz de servicio, comienza por crear una interfaz que herede de RemoteService.

```
public interface MiServicio extends RemoteService {
public String miMetodo(String s);
}
```

Ésta interfaz **síncrona** es la versión definitiva de la declaración de tus servicios. Cualquier implementación de este servicio en el lado del servidor debe heredar a RemoteServiceServlet e implementar ésta interfaz de servicio.

```
public class ImplMiServicio extends RemoteServiceServlet implements
MiServicio {

public String miMetodo(String s) {
// Las cosas a realizar en el servidor.
return s;
}

}
```

Interfaces Asíncronas

Antes de que puedas intentar realizar una invocación remota desde el cliente, debes crear otra interfaz, una asíncrona, basada en tu interfaz de servicio original. Continuando con el ejemplo de arriba sería algo así...

```
interface MiServicioAsync {
public void miMetodo(String s, AsyncCallback callback);
}
```

La naturaleza de la invocación de métodos asíncronos requiere que *quien realice la invocación* pase un objeto callback que pueda ser notificado cuando la llamada asíncrona esté completa, ya que *quien realice la invocación* no puede ser bloqueado hasta que la llamada esté completa. Por la misma razón, los métodos asíncronos no tienen tipo de retorno; siempre deben retornar `void`. Después que una llamada asíncrona es hecha, la información de retorno a *quien hizo la invocación* es hecha a través de un objeto callback.

La relación entre un servicio y su contraparte asíncrona es sencilla:

- Si una interfaz de servicio es llamada `com.ejemplo.cal.client.SpellingService`, entonces la interfaz asíncrona debe ser llamada `com.ejemplo.cal.client.SpellingServiceAsync`. La interfaz asíncrona debe estar en el mismo paquete y tener el mismo nombre, pero con el sufijo **Async**.

- Para cada método en tu interfaz de servicio,

```
public TipoRetorno metodoNombre(TipoParametro param1, TipoParametro
param2);
```

un método asíncrono debe ser definido de la siguiente manera:

```
public void metodoNombre (TipoParametro param1, TipoParametro param2,
AsyncCallback callback);
```

Ver AsyncCallback para detalles adicionales sobre cómo implementar un objeto callback asíncrono.

e) Implementando servicios

Cada servicio finalmente necesita realizar algunos procesos con el fin de responder peticiones a los clientes. Cada proceso del lado del servidor ocurre en la implementación del servicio, que está basada en la conocida arquitectura de los servlets.

Una implementación de servicio debe heredar de RemoteServiceServlet y debe implementar la interfaz de servicio asociada. Puedes notar que la implementación del servicio no implementa la versión asíncrona de la interfaz de servicio.

Cada implementación de servicio es finalmente un servlet, pero en vez de heredar de HttpServlet, hereda de RemoteServiceServlet. RemoteServiceServlet maneja automáticamente la serialización e invoca el método requerido en tu implementación de servicio.

Testeando servicios durante el Desarrollo

Para cargar automáticamente tu implementación de servicio, usa la etiqueta dentro de tu modulo XML. El shell de desarrollo de Google WebToolkit incluye una versión incrustada de Tomcat que actúa como un contenedor de servlets en tiempo de desarrollo para pruebas.

Servicios en Producción

En producción, puedes usar cualquier contenedor de servlets que sea apropiado para tu aplicación. Solamente necesitas asegurarte que el código del cliente está configurado para invocar el servicio usando la URL para la cual tu servlet es mapeado por la configuración de `web.xml`. Ver ServiceDefTarget para más información.

f) Realizando una invocación

El proceso de realizar un RPC desde un cliente involucra siempre los mismos pasos.

1. Instanciar la interfaz de servicio usando GWT.create().
2. Especificar un URL para el servicio usando ServiceDefTarget.
3. Crear un objeto callback asíncrono para ser notificado cuando el RPC se haya completado.
4. Realizar la llamada/invocación.

Ejemplo

Supón que quieres llamar un método sobre una interfaz de servicio definida así:

```
public interface MiServicioEmail extends RemoteService {
void BandejaEntrada(String nombreUsuario, String password);
}
```

Y su correspondiente interfaz asíncrona se ve algo así:

```
public interface MiServicioEmailAsync {
void BandejaEntrada (String nombreUsuario, String password,
AsyncCallback callback);
}
```

La llamada/invocación desde el lado del cliente tendría que verse así:

```
public void menuCommandBandejaEntrada() {
// (1) Crear el proxy del cliente. Nota que aunque creas la
// interfaz de servicio adecuada, repartes el resultado la versión
// asíncrona de la interfaz. El reparto es siempre seguro ya que
// el proxy generado implementa la interfaz asíncrona automáticamente
//
MiServicioEmailAsync servicioEmail = (MiServicioEmailAsync)
GWT.create(MiServicioEmail.class);

// (2) Especificar la URL que nuestra implementación de
// servicio está corriendo.
// Nota que la URL debe residir sobre el mismo dominio y puerto
// desde la que la página fue servida.
//
ServiceDefTarget endpoint = (ServiceDefTarget) servicioEmail;
String URLmodulo = GWT.getModuleBaseURL() + "email";
endpoint.setServiceEntryPoint(URLmodulo);

// (3) Crear un objeto callback asíncrono para manejar el resultado.
//
AsyncCallback callback = new AsyncCallback() {
public void onSuccess(Object result) {
// realizar los procesos que sean necesarios
// por ejemplo actualizar la interfaz de usuario
}

public void onFailure(Throwable caught) {
// realizar los procesos necesarios si falló la operación
}
};

// (4) Realizar la llamada. El control de flujo continuará
// inmediatamente y luego 'callback' será invocado cuando
// el RPC se haya completado.
//
servicioEmail.BandejaEntrada(vNombreUsuario, vPassword, callback);
}
```

Es seguro guardar en caché los servicios instanciados, para evitar crearlos en posibles llamadas/invocaciones subsecuentes.

Entradas relacionadas

- RemoteService, ServiceDefTarget, GWT.create(Class)

Bibliografía:

Google Web Toolkit Project

GWT Powered