Familiarize-se com o Google Web Toolkit

JavaScript do Google

O engenhoso Google Web Toolkit cria rapidamente aplicações JavaScript otimizadas.

por Dan Frost

JavaScript é o responsável por tirar dias, semanas, talvez até meses dos programadores. O recente surgimento de *frameworks* nessa linguagem – e sua crescente estabilidade – ajudam a melhorar esse quadro. O *Google Web Toolkit* (GWT) [1] parece o próximo estágio evolutivo no desenvolvimento em JavaScript: em vez de escrever em JavaScript, pode-se escrever em Java.

O GWT é um ambiente para criação de aplicações JavaScript otimizadas compatíveis com múltiplos navegadores. Com o GWT, criam-se aplicações JavaScript programando-se em Java e compilando-se o código para HTML, CSS e JavaScript altamente otimizados. Por mais que se aprecie trabalhar com os intricados probleminhas multinavegador do JavaScript, há um ponto em que a paciência acaba. O GWT apareceu exatamente antes de muitos programadores alcançarem esse ponto.

De perto

O GWT fornece uma biblioteca de layouts, elementos de formulário e outros componentes para a criação de aplicações web. Em vez de adicionar código JavaScript/AJAX sobre HTML e CSS puros, pode-se usar componentes Java de nível mais alto que o GWT compila em JavaScript adequado para todos os navegadores e que provavelmente não precisará de depuração.

Entretanto, não é preciso abrir mão completamente da programação em JavaScript. O JavaScript ainda tem seu lugar, e esse artigo mostrará como expor partes da aplicação criada com o GWT ao JavaScript, criando assim uma API adequada.

Alguns benefícios do GWT são:

- compatibilidade com múltiplos navegadores – ao usar componentes de alto nível do GWT, não se gasta tanto tempo na depuração;
- desempenho – o código JavaScript obtido é otimizado, assim como tempos de download menores;
- exposição da API ao JavaScript – é possível migrar para o GWT seção por seção, usando a API JavaScript já existente;
- depuração e desenvolvimento – utiliza-se uma linguagem mais forte que o JavaScript – com melhores ferramentas de depuração.

O fluxo de adoção do GWT costuma partir de uma aplicação já existente e criada originalmente em JavaScript, HTML e CSS. Essa solução se mostra difícil e trabalhosa para depurar, e o GWT começa a parecer uma solução bastante atraente; porém, como plugá-lo nas áreas problemáticas? Este artigo mostra como usar o GWT para criar e integrar componentes JavaScript com páginas web e aplicações já existentes.

Primeiros passos

Primeiramente, baixe uma versão apropriada do GWT para seu sistema operacional [2]. Esse download contém um conjunto de exemplos, o compilador Java-para-JavaScript e ferramentas para criação de novas aplicações e execução de testes. Em seguida, descompacte o arquivo e use o `applicationCreator`:

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

A ferramenta `applicationCreator` cria um novo projeto do GWT para criação de aplicações em JavaScript.

O `projectCreator`, no mesmo diretório, é usado para criar projetos do GWT para edição no *Eclipse*, mas não é necessário depender desse IDE.

Inicialmente, crie um projeto simples:

```
$ ./applicationCreator
➥-out ~/meuprojeto client.
➥com.mycompany.MyApp
```

O `applicationCreator`, com isso, cria os arquivos em `~/meuprojeto/`:

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

Esse pequeno número de arquivos é bom porque significa que o que se vê é a base do trabalho, em vez de haver dezenas de diretórios com conteúdo de relevância duvidosa.

Agora, abra `client/myapp.java`, que contém a classe Java que o GWT compilará para código JavaScript. Este artigo se limita a uma classe, mas é possível recriar o código para usar outras, como se faria com qualquer outro projeto. Para iniciar a shell do GWT, digite o comando:

```
$ ./MyApp-shell
```

Exemplo 1: Compilação com o 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
```

Exemplo 2: Conteúdo de com.mycompany.MyApp

```
548CDF11D6FE9011F3447CA200D7FB7F.cache.png
9DA92932034707C17CFF15F95086D53F.cache.png
A84A8EF7341E8139F58DC5FC2AD52F22.cache.html
MyApp.css
MyApp.html
clear.cache.gif
com.mycompany.MyApp.nocache.js
gwt/
   history.html
   hosted.html
```

Isso inicia o modo hospedado, ou *hosted mode* (**figura 1**), que consiste em um navegador dedicado para o ambiente de desenvolvimento, que recompila código Java para JavaScript a cada atualização. Quando você altera o arquivo Java e atualiza a shell do GWT, os resultados são mostrados imediatamente, eliminando a necessidade de compilar o Java a cada vez. Mantenha o shell do GWT aberto na próxima etapa.

A aplicação criada pelo GWT é rápida e simples, e mostra alguns componentes padrão de páginas web: botões, imagens, caixas e caixas de diálogo. Em seguida, abra o arquivo `MyApp.java` e altere a linha:

```
dialogBox.setText("Bem vindo ao
➥GWT!");
```

para:

```
dialogBox.setText("Bem vindo ao
➥GWT - está gostando?");
```

Agora atualize o shell do GWT, clique no botão e veja o Java ser compilado para JavaScript imediatamente. Evidentemente, o modo hospedado só é útil durante o desenvolvimento, então é possível usar o `compile` para compilar a aplicação para um conjunto de arquivos JS,

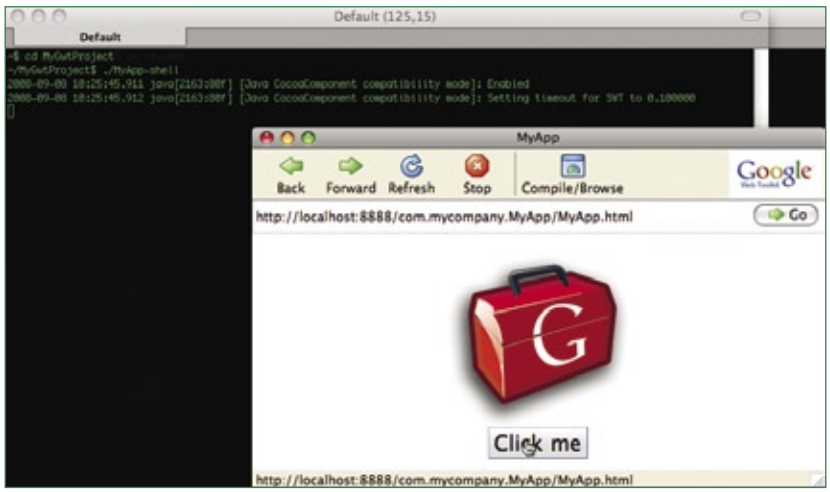

Figura 1 Entrando no modo hospedado do GWT.

Exemplo 3: MyApp.java

```
01 public class MyApp implements EntryPoint {
02    //Este é o método do ponto de entrada.
03    public void onModuleLoad() {
04       Window.alert("Ola");
05    }
06 }
```

Exemplo 4: Função a acrescentar

```
01 public void openDialogBox() {
02    final DialogBox dialogBox = new DialogBox();
03    dialogBox.setText("Esta é minha caixa de diálogo simples");
04    dialogBox.setAnimationEnabled(true);
05    dialogBox.center();
06    dialogBox.show();
07 }
```

HTML, CSS e imagens (**exemplo 1**). A aplicação depois é compilada num novo diretório, `www/com.mycompany.MyApp/`, que contém os arquivos mostrados no **exemplo 2**.

Com isso, a aplicação está completamente auto-contida; é possível mover o diretório `www/com.mycompany.MyApp/` para outro local – por exemplo, para dentro de uma aplicação web já existente:

```
$ mv www/com.mycompany.MyApp   \
/caminho/da/outra/aplicação/
```

O uso do GWT já deve começar a fazer sentido: em vez de compilar o Java para JS toda vez que for terminado um recurso ou consertada uma falha, pode-se trabalhar no modo hospedado até a aplicação estar 90% terminada, e só então compilá-la para JavaScript.

Dentro do Java

A classe Java implementa o `EntryPoint/onModuleLoad`, que o GWT aciona a cada carregamento de página. Então, ele pode ser encarado como o `onload` em páginas HTML ou o `dom ready` em bibliotecas como as *MooTools*. Isso pode ser verificado com um exemplo bem pequeno; edite o arquivo `MyApp.java` para que a classe contenha somente as linhas mostradas no **exemplo 3**.

Intuitivamente, `Window.alert()` está chamando a função JavaScript de alerta. Se for iniciado o modo hospedado usando o `MyApp-shell`, será possível verificar que não há muitos acontecimentos, mas já é possível constatar como a experiência do usuário em JavaScript se enquadra em torno do código Java.

Diálogo simples

Em seguida, tente criar uma caixa de diálogo simples que seja carregada a partir de uma URL AJAX. Lembre-se de importar todo o conteúdo do pacote cliente do GWT:

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

Substitua o conteúdo do `onModuleLoad` por:

```
openDialogBox();
```

e depois acrescente a função do **exemplo 4**.

Se isso for aberto no modo hospedado, a caixa de diálogo aparecerá quando a página abrir, o que é uma demonstração interessante mas pouco útil. Em vez disso, é melhor conseguir chamar a caixa de diálogo a partir de qualquer parte da aplicação. Para isso, é preciso expor parte da aplicação Java para JavaScript usando a *JavaScript Native Interface* (JSNI).

Primeiro, crie uma função que se declare "nativa" e que de fato inicialize uma "API" do JavaScript, como a do **exemplo 5**.

Por último, em `onModuleLoad`, substitua `openDialogBox()` por `initJavaScriptAPI()`:

```
initJavaScriptAPI(this);
```

Ao ser atualizado o modo hospedado, nada será visível quando a aplicação for carregada, pois ela declara somente a API JavaScript – só se abre uma janela quando a função JavaScript `openDialog()` for chamada. Para ver isso em funcionamento, acrescente a seguinte linha na seção `<body>` de `public/MyApp.html`:

Exemplo 5: Usando funções nativas

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

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

Depois, atualize o modo hospedado, clique no link e veja o diálogo se abrir. Apesar de esse exemplo ser básico, ele demonstra algo útil: agora, você já pode criar recursos complexos e multinavegador no GWT e chamá-los a partir das aplicações JavaScript já existentes.

Aplicações AJAX

A caixa de diálogo ainda é apenas um demo interessante, e não algo realmente útil; é possível deixá-la mais interessante acrescentando AJAX entre um servidor AJAX já existente e o GWT.

Um requisito típico é que a caixa de diálogo carregue seu conteúdo por AJAX, o que é fácil de alcançar, modificando a classe personalizada. Adicione o código do **exemplo 6** ao final de `openDialogBox()`.

Exemplo 6: Adição do 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       // Impossível conectar ao servidor (pode ser timeout,
➥violação de SOP etc.)
08       dialogBox.setText( "Lamento - Não consegui carregar
➥o HTML");
09    }
10
11    public void onResponseReceived(Request request,
➥Response response) {
12       if (200 == response.getStatusCode()) {
13          dialogBox.setText( response.getText() );
14       } else {
15          // Trata o erro. Pode obter o texto de status a partir
de response.getStatusText()
16          dialogBox.setText( "Lamento - recebi a resposta mas
➥não a entendi!");
17       }
18    }
19    );
20 } catch(Exception e) { }
```

Exemplo 7: Modificação de InitJavaScriptAPI

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

Em seguida, use `MyApp-compile` para instalá-lo em uma aplicação já existente. Serão necessárias algumas páginas web rodando localmente – suponhamos que você já tenha um servidor LAMP em execução.

Depois, compile a aplicação e copie o diretório `www/` para a aplicação já existente:

```
$ ./MyApp-compile
$ mv -r www /caminho/da/aplicação/
➥gwt-www
$ touch /caminho/da/aplicação/gwt/
➥www/AjaxServer.php
```

Por último, crie um arquivo chamado `AjaxServer.php` e acrescente o seguinte conteúdo:

```
<php
echo "Olá, Mundo. Aqui fala o
➥servidor AJAX executado pelo GWT
➥mas acionado a partir de uma
➥chamada nativa do JavaScript!";
➥?>
```

Testando

Para testar o novo recurso de AJAX, abra o arquivo `MyApp.html` de dentro da aplicação e clique no link. A API JavaScript significa que é possível chamar recursos do GWT de dentro da aplicação JS pré-existente, ao passo que usar o servidor AJAX significa que o GWT é capaz de se integrar aos recursos AJAX atuais. Porém, a URL AJAX é *hardcoded*, então é necessário colocá-la numa variável passada do JavaScript para Java.

Primeiro, modifique o conteúdo de `initJavaScriptAPI` (veja o **exemplo 7**). O argumento `url` na **linha 1** é um parâmetro JavaScript que será convertido para uma variável Java de tipo `java.lang.string` e passado para `openDialogBox`.

Depois, modifique `openDialogBox` para aceitar o argumento:

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

e depois modifique a requisição para usar essa variável:

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

Agora, compile isso, mova os arquivos para a sua aplicação e depois adicione algums URLs às chamadas de funções JS para conseguir chamar URLs que já existem:

```
<a href="javascript:openDialog('/
➥AjaxServer1.php');">Abrir a caixa
➥ de diálogo do GWT</a> <a
➥ href="javascript:openDialog('/
➥AjaxServer2.php');">Abrir outra
➥ caixa de diálogo</a>
```

Na última etapa, coloque o recurso do GWT na aplicação web incluindo um arquivo JavaScript. Adicione a *tag* do script ao cabeçalho HTML, ajustando o termo `src=".."` para que aponte para o diretório adequado:

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

Depois, em algum local do template, acrescente os links que acionam o JavaScript:

```
<a href="javascript:openDialog('/
➥caminho/do/ajax.php');">Abrir meu
➥ servidor Ajax</a>
```

Como teste, é possível incluir esse link no *WordPress*, no seu CMS ou em qualquer outra página web.

Otimização

É fácil perceber que o JavaScript se torna mais lento conforme a aplicação cresce. Isso demanda longas horas de otimização manual de código JavaScript, o que leva a pesadelos de depuração, principalmente por JavaScript ser fracamente tipada.

Migração

A mudança de códigos leves em JavaScript para o robusto e pesado Java não é trivial, mas é possível migrar lentamente, ou migrar apenas as áreas mais problemáticas da aplicação. Para migrar recursos do JavaScript para o GWT, é preciso começar com uma API bastante sólida. Normalmente, isso significa chamar somente um ou dois recursos na página web.

Comece criando os componentes e recursos no GWT com uso dos componentes e recursos do GWT em vez de HTML e JavaScript puros. É importante usar componentes de alto nível que forneçam a segurança de funcionarem em qualquer navegador.

Em seguida, exponha partes específicas do componente GWT para JavaScript com uso das funções "nativas". Elas provavelmente serão parecidas com a API que já estava em uso, o que é bom para preservar a retrocompatibilidade. Por último, basta incluir o JavaScript gerado pelo GWT e retirar o código antigo da página web. ■

Mais informações

[1] Google Web Toolkit: http://code.google.com/webtoolkit/

[2] Download do GWT: http://code.google.com/webtoolkit/download.html

[3] Gwt-fx (animação básica do GWT): http://code.google.com/p/gwt-fx/

[4] GWT Ext JS (framework para o GWT): http://extjs.com/products/gxt/

[5] GWT on Rails: http://code.google.com/p/gwt-on-rails/