





uma página com nada de especial, mas ela está assim porque nós ainda não adicionamos nada de especial nela, mas agora que tudo está pronto e configurado, podemos iniciar o desenvolvimento do nosso sistema de gerenciamento de contatos.

O código que gera esta página é extremamente simples e demonstra como também é simples trabalhar com o GWT:

#### Listagem 1. Código inicial gerado pelo GWT

```
public class GWTutorial implements EntryPoint {
    /**
     * This is the entry point method.
     */
    public void onModuleLoad() {
        final Button button = new Button("Click me");
        final Label label = new Label();

        button.addClickListener(new ClickListener() {
            public void onClick(Widget sender) {
                if (label.getText().equals(""))
                    label.setText("Hello World!");
                else
                    label.setText("");
            }
        });

        // Assume that the host HTML has elements defined whose
        // IDs are "slot1", "slot2". In a real app, you probably would not want
        // to hard-code IDs. Instead, you could, for example, search for all
        // elements with a particular CSS class and replace them with widgets.
        //
        RootPanel.get("slot1").add(button);
        RootPanel.get("slot2").add(label);
    }
}
```

Todas as aplicações GWT se iniciam em uma classe que implementa a interface `EntryPoint`, que define o método `onModuleLoad()` chamado quando a aplicação começa a executar para no cliente. É aqui que nós fazemos a inicialização da interface da aplicação para ser mostrada no navegador. Como já havia sido dito, todo o código escrito é Java puro, primeiro é instanciado um objeto do tipo `Button`, com o texto “Click Me”, depois é instanciado um `Label` sem texto. Adicionamos um “listener” para o clique do botão, que coloca ou limpa o texto do `Label` e finalmente adicionamos os objetos criados a interface do navegador.

A parte mais estranha do código é exatamente o final, onde fazemos uma chamada ao método estático `get()` da classe `RootPanel`. O `RootPanel` simboliza a “página” atual onde a aplicação está executando e o `String` passado como parâmetro é o valor do “id” HTML das tags onde esse conteúdo vai ser adicionado. Nos arquivos gerados durante a chamada ao “applicationCreator” há um arquivo HTML que é exatamente o arquivo criado pelo projeto para conter a nossa aplicação. A parte onde essas tags são declaradas é a seguinte:

#### Listagem 2 – /src/org/maujr/gwt/tutorial/public/GWTutorial.html - Tags HTML onde os componentes vão ser adicionados

```
<table align=center>
  <tr>
    <td id="slot1"></td><td id="slot2"></td>
  </tr>
</table>
```

Então, a chamada ao método `get()` do `RootPanel` retorna uma referência para as tags HTML referenciadas nesse arquivo e identificadas pelo “id” correspondente. O botão é colocado a esquerda, no “slot1” e o `Label` é colocado a direita no “slot2”.

## Desenvolvendo a aplicação

Para iniciar a nossa aplicação, vamos fazer algumas modificações no código gerado pelo GWT. Nós temos que adicionar espaços para colocar a lista de cidades de destino, a lista das cidades de origem e a tabela que vai mostrar o resultado da nossa busca. Para fazer isso, nós adicionamos algumas tags HTML onde antes havia a tabela que o exemplo mostrava a mensagem e o botão. Vejamos o código:



```
// métodos get/set e construtores
}
```

Com a interface base do serviço e o objeto que ele retorna definidos corretamente, nós precisamos definir uma “interface assíncrona” para a execução do serviço no cliente. A definição da interface assíncrona é necessária porque a invocação ao serviço utilizando AJAX acontece de forma assíncrona, para o cliente, a aplicação não para de executar, a chamada ao serviço acontece em background, então é necessário definir a interface assíncrona que vai ser a interface real do serviço no cliente. Vejamos a interface assíncrona do nosso serviço:

### Listagem 6 – Interface assíncrona do serviço de busca de viagens

```
package org.maujr.gwt.tutorial.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface BuscaViagensServiceAsync {

    public void getViagens( String[] origens, String[] destinos, AsyncCallback callback );

}
```

A interface assíncrona do serviço tem as declarações de métodos praticamente iguais as da interface do serviço definida anteriormente, mas os métodos de serviço agora devem retornar “void” e o último parâmetro deve ser um objeto do tipo **com.google.gwt.user.client.rpc.AsyncCallback**, que é o objeto que vai funcionar como um “listener” do evento de execução do serviço. AsyncCallback é uma interface que define dois métodos, “onSucess()” que é chamado quando o método executa corretamente e tem como parâmetro o resultado da invocação do serviço (por isso o método da interface assíncrona retorna “void”) e “onFailure()” que é chamado quando a chamada do serviço não acontece de forma correta e recebe como parâmetro um objeto do tipo Throwable com informações sobre o erro acontecido. O nome da interface assíncrona deve ser o mesmo nome da interface real do serviço, adicionando o sufixo “Async” e as duas devem estar dentro do mesmo pacote.

Com as interfaces do serviço definidas, vamos implementá-lo no servidor. Para fazer isso, criamos um objeto que estenda **com.google.gwt.user.server.rpc.RemoteServiceServlet** e implemente a interface real do serviço (a BuscaViagensService). Esse objeto deve ficar dentro do pacote “server” no mesmo nível do pacote “client” e “public” da sua aplicação, para que o GWT saiba que esse objeto não deve ser transformado em JavaScript, pois ele pode vir a fazer uso de várias APIs do Java que não estão disponíveis em JavaScript. Vejamos a nossa implementação do serviço:

### Listagem 7 – Implementação do serviço de busca de viagens no servidor

```
public class BuscaViagensServiceImpl extends RemoteServiceServlet implements
    BuscaViagensService {

    private static List<Viagem> VIAGENS = new ArrayList<Viagem>();

    static {

        VIAGENS.add(new Viagem(CIDADES[0], CIDADES[1], EMPRESAS[0] ));
        VIAGENS.add(new Viagem(CIDADES[7], CIDADES[6], EMPRESAS[1] ));
        VIAGENS.add(new Viagem(CIDADES[3], CIDADES[5], EMPRESAS[2] ));
        VIAGENS.add(new Viagem(CIDADES[2], CIDADES[4], EMPRESAS[3] ));
        VIAGENS.add(new Viagem(CIDADES[7], CIDADES[0], EMPRESAS[4] ));
        VIAGENS.add(new Viagem(CIDADES[6], CIDADES[3], EMPRESAS[0] ));
        VIAGENS.add(new Viagem(CIDADES[5], CIDADES[0], EMPRESAS[1] ));
        VIAGENS.add(new Viagem(CIDADES[0], CIDADES[2], EMPRESAS[2] ));
        VIAGENS.add(new Viagem(CIDADES[2], CIDADES[7], EMPRESAS[2] ));

    }

    /**
     * Comentário para <code>serialVersionUID</code>
     */
    private static final long serialVersionUID = -7912694211530826639L;

    /**
     * (não-Javadoc)
     *
     * @see org.maujr.gwt.tutorial.client.BuscaViagensService#getViagens(java.lang.String,
     *      java.lang.String, java.lang.String[])
     */
    public Viagem[] getViagens(String[] origens, String[] destinos) {

        boolean compararOrigens = origens != null && origens.length > 0 ? true : false;
```

```

boolean compararDestinos = destinos != null && destinos.length > 0 ? true : false;
Set<Viagem> resultado = new HashSet<Viagem>();

for (Viagem viagem : VIAGENS) {

    boolean adicionarOrigem = false;
    boolean adicionarDestino = false;
    if (!compararOrigens && !compararDestinos) {
        resultado.add(viagem);
    } else {

        if (compararOrigens) {
            adicionarOrigem = contains( viagem.getOrigem(), origens );
        }

        if (compararDestinos) {
            adicionarDestino = contains( viagem.getDestino(), destinos );
        }

        if (adicionarOrigem || adicionarDestino)
            resultado.add(viagem);
    }
}

Iterator<Viagem> iterator = resultado.iterator();
Viagem[] array = new Viagem[resultado.size()];

for (int x = 0; iterator.hasNext(); x++) {
    array[x] = iterator.next();
}

return array;
}

private static boolean contains(String nome, String[] array) {

    for ( String valor : array ) {

        if ( nome.equalsIgnoreCase(valor) ) {
            return true;
        }

    }

    return false;
}
}

```

Para simplificar a execução e entendimento do exemplo, o serviço faz a busca em um conjunto de objetos definido estaticamente no Servlet e não em um banco de dados, mas isso não impede que você altere o exemplo para fazer a pesquisa de qualquer outra forma ou em qualquer outro lugar. O intuito do exemplo é apenas mostrar o que pode ser feito utilizando as funcionalidades da invocação remota de serviços do GWT, dentro dessa classe, como você tem acesso a todos os “poderes” do Java, vai poder utilizar o que quiser, bastando apenas que você retorne o tipo de objeto que o serviço espera receber.

## Invocando o serviço do cliente

Com o nosso serviço definido e implementado, nós precisamos agora fazer com que ele seja invocado na nossa aplicação cliente. Para fazer isto e adicionar os componentes necessários na nossa aplicação, nós vamos fazer algumas alterações na nossa classe que carrega a aplicação no cliente. Vejamos a listagem 8 com a implementação da classe:

### Listagem 8 – Implementação do ponto de entrada na aplicação

```

public class GWTTutorial implements EntryPoint, ClickListener, AsyncCallback {

    private BuscaViagensServiceAsync service;

    private VerticalPanel destinos;

    private VerticalPanel origens;
}

```

```

private FlexTable table;

/**
 * This is the entry point method.
 */
public void onModuleLoad() {

    this.table = new FlexTable();
    this.destinos = new VerticalPanel();
    this.origens = new VerticalPanel();

    for ( int x = 0; x < 8; x++ ) {
        this.destinos.add( createCheckBox( Constants.CIDADES[x], this ) );
        this.origens.add( createCheckBox( Constants.CIDADES[x], this ) );
    }

    this.service = (BuscaViagensServiceAsync) GWT.create(BuscaViagensService.class);

    ServiceDefTarget target = (ServiceDefTarget) service;
    target.setServiceEntryPoint( GWT.getModuleBaseURL() + "viagens" );

    RootPanel.get("destinos").add(this.destinos);
    RootPanel.get("origens").add(this.origens);
    RootPanel.get("viagens").add(this.table);

    String[] arrayVazio = {};

    this.service.getViagens(arrayVazio, arrayVazio, this);

}

/* (não-Javadoc)
 * @see
 * com.google.gwt.user.client.ui.ClickListener#onClick(com.google.gwt.user.client.ui.Widget)
 */
public void onClick(Widget sender) {

    List origensSelecionadas = new ArrayList();
    List destinosSelecionados = new ArrayList();

    Iterator origensIterator = this.origens.iterator();
    while (origensIterator.hasNext()) {

        CheckBox checkBox = (CheckBox) origensIterator.next();

        if (checkBox.isChecked()) {
            origensSelecionadas.add( checkBox.getText() );
        }

    }

    Iterator destinosIterator = this.destinos.iterator();
    while (destinosIterator.hasNext()) {

        CheckBox checkBox = (CheckBox) destinosIterator.next();

        if (checkBox.isChecked()) {
            destinosSelecionados.add( checkBox.getText() );
        }

    }

    String[] origensArray = new String[origensSelecionadas.size()];
    String[] destinosArray = new String[destinosSelecionados.size()];

    for ( int x = 0; x < origensArray.length; x++ ) {
        origensArray[x] = (String) origensSelecionadas.get(x);
    }

    for ( int x = 0; x < destinosArray.length; x++ ) {
        destinosArray[x] = (String) destinosSelecionados.get(x);
    }

    this.service.getViagens(origensArray, destinosArray, this);

}

```

```

private void carregarTabela(Viagem[] viagens) {

    while ( this.table.getRowCount() > 0 ) {
        this.table.removeRow(0);
    }

    this.table.setHTML(0, 0, "<strong>Empresa</strong>");
    this.table.setHTML(0, 1, "<strong>Origem</strong>");
    this.table.setHTML(0, 2, "<strong>Destino</strong>");

    for ( int x = 0; x < viagens.length; x++ ) {
        Viagem viagem = viagens[x];
        this.table.setText(x + 1, 0, viagem.getEmpresa());
        this.table.setText(x + 1, 1, viagem.getOrigem());
        this.table.setText(x + 1, 2, viagem.getDestino());
    }

}

/* (não-Javadoc)
 * @see com.google.gwt.user.client.rpc.AsyncCallback#onFailure(java.lang.Throwable)
 */
public void onFailure(Throwable caught) {

    // fazer o tratamento de erro

    Window.alert("Ocorreu um erro: " + caught);

}

/* (não-Javadoc)
 * @see com.google.gwt.user.client.rpc.AsyncCallback#onSuccess(java.lang.Object)
 */
public void onSuccess(Object result) {

    carregarTabela((Viagem[]) result);

}

private static CheckBox createCheckBox(String text, ClickListener listener) {

    CheckBox checkBox = new CheckBox(text);
    checkBox.setText(text);

    checkBox.addClickListener(listener);

    return checkBox;

}

}

```

A classe GWTTutorial, que é o ponto de entrada do cliente na nossa aplicação, ganhou algumas novas responsabilidades. Ela agora guarda as referências para os componentes visuais que nós vamos utilizar na página e implementa as interfaces ClickListener, para responder aos cliques nos checkboxes, e AsyncCallback para responder a execução da chamada remota ao serviço. Em uma aplicação maior, você poderia dividir melhor as responsabilidades das classes e criar objetos especificamente para implementar estas interfaces.

Na nossa classe, temos dois VerticalPanels, que são componentes do tipo painel, que são utilizados para organizar os componentes que são inseridos neles de alguma forma. No caso dos VerticalPanels eles fazem com que os componentes adicionados sejam colocados em uma lista vertical de componentes. O outro componente que está na classe é a FlexTable, que é uma tabela que pode ser redimensionada dinamicamente, ela vai ser utilizada para mostrar o resultado das consultas.

Todos os objetos são inicializados na chamada do método “onModuleLoad()”, é nele que nós criamos os componentes e criamos o objeto que vai fazer as chamadas ao serviço remoto. Para criar um objeto que faz a chamada ao serviço, nós utilizamos o método estático “create()” da classe “GWT”, passando como parâmetro o objeto class da interface real do serviço. O objeto retornado é um objeto que implementa a interface assíncrona do serviço (no nosso caso, a interface BuscaViagensServiceAsync).

Após retornado o objeto, nós ainda temos que fazer um “cast” dele para o tipo **com.google.gwt.user.client.rpc.ServiceDefTarget** e dizer qual é a URL na qual o serviço deve ser invocado. Essa URL é o mapeamento do nosso servlet que implementou o serviço remoto e essa URL deve ser absoluta. Para não ter que definir diretamente o caminho do contexto, nós utilizamos o método estático **GWT.getModuleBaseURL()**, que retorna o caminho absoluto até a aplicação definida (com “/” no final) e fazemos a concatenação com a URL do mapeamento do servlet na aplicação, que nesse caso vai ser “/viagens”.

Para testar se tudo está funcionando corretamente, nós podemos declarar o servlet do serviço no arquivo de configuração do módulo do GWT, pois como ele mesmo tem um servidor web para testes, nós podemos utilizar o seu próprio ambiente de testes para executar a aplicação. Vejamos como deve ficar o arquivo de configuração:

### Listagem 9 – GWTTutorial.gwt.xml - Arquivo de configuração da aplicação

```
<module>
  <!-- Inherit the core Web Toolkit stuff.          -->
  <inherits name='com.google.gwt.user.User' />
  <!-- Specify the app entry point class.          -->
  <entry-point class='org.maujr.gwt.tutorial.client.GWTTutorial' />
  <servlet
    path='/viagens'
    class='org.maujr.gwt.tutorial.server.BuscaViagensServiceImpl' />
</module>
```

Nós definimos o servlet e o seu mapeamento e agora já estamos prontos para executar a aplicação, ao executar o arquivo GWTTutorial-shell.cmd (ou .sh) você deve ver uma tela como a seguinte:

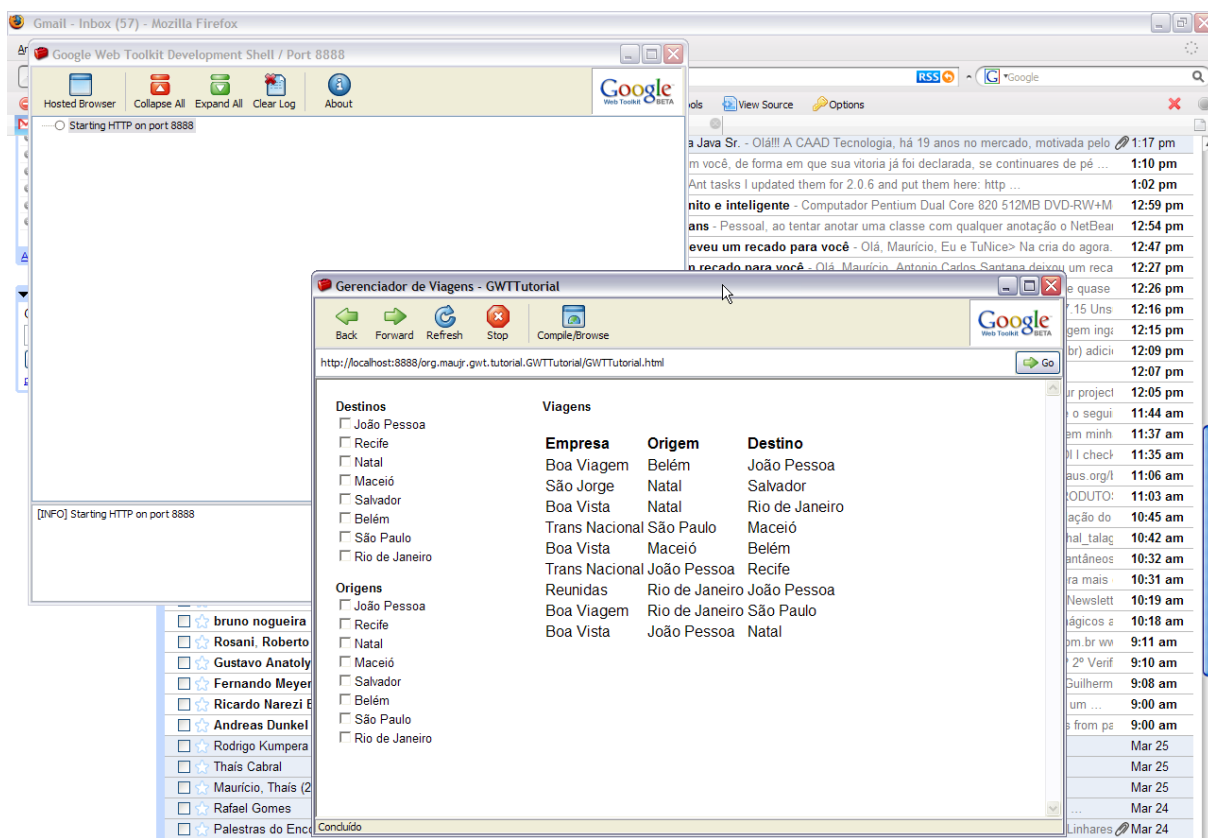


Imagem 2 – Aplicação GWT executando no navegador de testes

A aplicação executa normalmente, acessando dados no servidor e sem fazer refresh na página inteira, atualizando apenas os campos que precisam ser atualizados, diminuindo o consumo de banda e de processamento no servidor, pois apenas os dados que são necessários são requisitados. Temos então a nossa aplicação implementada e executando normalmente no servidor.

## Implantando a aplicação em outro servidor

Para implantar a sua aplicação em outro servidor, você deve empacotar todos os arquivos .class gerados (de todos os pacotes) e colocar no classpath da sua aplicação, mapear o servlet que implementa um serviço remoto e gerar os arquivos JavaScript necessários executando o script GWTTutorial-compile.cmd (ou .sh). Ele vai gerar os arquivos dentro de uma pasta “www” na pasta do projeto. Esses arquivos contidos nesta pasta devem ser colocados dentro da pasta raiz da aplicação web. Para executar a aplicação basta chamar o arquivo GWTTutorial.html.

### Para Saber Mais

Google Web Toolkit – Página oficial do projeto - <http://code.google.com/webtoolkit/>

Blog da equipe do GWT - <http://googlewebtoolkit.blogspot.com/>

### Considerações Finais

Mesmo tendo uma abordagem um relativamente diferente na criação de aplicações AJAX, a simplicidade do uso do GWT faz dele uma opção interessante especialmente para aqueles que desejam estender as funcionalidades de aplicações já existentes com o uso de AJAX. Mas essa abordagem relativamente diferente também complica o seu uso, pois editar os arquivos de script gerados para adicionar novas bibliotecas ao classpath ou gerar versões compiladas das aplicações fica mais difícil conforme a quantidade de dependências aumenta.

Mas esses problemas são apenas ferramentais e a melhora do suporte a compilação e geração do JavaScript que deve vir com os plugins que já estão sendo desenvolvidos devem simplificar ainda mais a criação de aplicações com o GWT. Agora só não espere pra ver, senão você pode perder a parada.

**Maurício Linhares de Aragão Junior** ([mauricio.linhares@gmail.com](mailto:mauricio.linhares@gmail.com)) é graduando em **Desenvolvimento de Sistemas para a Internet** e Comunicação Social, com Habilitação em **Jornalismo**, desenvolvedor da **Phoebus Tecnologia** (<http://www.phoebus.com.br/>), consultor e instrutor independente, instrutor parceiro da **Synapse Tech Cursos** (<http://stcursos.com/>), membro da equipe administrativa do **PBJUG** (<http://pbjug.org/>) e colaborador dos fóruns do **GUJ** (<http://guj.com.br/>).