Google Web Toolkit
Принимая на себя страдания Ajax

Эдд Бернед

Содержание

Глава 1
Вступление

Google Web Toolkit (GWT) был неожиданно открыт для публики 18 мая 2006 года на ежегодной конференции JavaOne в Сан-Франциско. Цель, которая стояла перед GWT, была очень простой: сделать разработку с помощью технологии Ajax проще за счет сокрытия несовместимостей браузеров от программиста и позволить разработчику работать в среде, подобной Java.

Провозглашение было одним из самых ярких событий конференции, и интересы продолжали расти. Разработчики использовали технологию GWT всюду: от игр до ипотечного калькулятора. Сообщество GWT опубликовало список из более чем 100 примеров, статей, виджетов и других ресурсов. Почему Google Web Toolkit?

Если вы когда-либо писали нестандартные Ajax-приложения, я уверен, что вы согласитесь с необходимостью сделать этот процесс проще. Если нет - понадобится немного пояснений.

1.1 Жизнь до GWT

Динамические web-приложения обычно пишутся на нескольких различных языках в несколько этапов. На клиентской стороне (браузерная часть), очевидно, есть HTML разметка, и имеется некоторый код на JavaScript для выполнения задач, подобных проверке данных на клиентской стороне и взаимодействию с моделью документа (DOM).

К сожалению, незначительные различия в языке JavaScript между браузерами вместе с более значительными отличиями в DOM, делают разработку таких клиентов как хождение по минному полю. Различные библиотеки, такие как Dojo и Prototype созданы для сглаживания грубых углов, но браузерное программирование на JavaScript до сих пор является черной магией. Некоторые разработчики отказываются от HTML и JavaScript в пользу Flash и других альтернатив.

На стороне сервера расположен серверный слой и, по желанию, слой данных. Платные web-серверы как Apache, Tomcat, Lighttpd и IIS взаимодействуют с логикой вашего приложения, написанной на Java, PHP, Ruby, C#, Klingon (а может и не Klingon), или других языках. JavaScript никем не используется на сервере, за исключением некоторых мазохистов. Сервисы данных обслуживаются базами данных, такими как MySQL, Oracle, Sql Server и т. Д. Часто сама база данных скрыта на объектно-реляционном слое, таком как Hibernate.

Несмотря на то, что эта архитектура довольно гибкая, она трудно управляемая. Средства разработки, подобные Ruby on Rails,

развиваются, чтобы уменьшить сложность серверной стороны. Другие средства, такие как Java Server Faces (JSF) и Microsoft Atlas пытаются стандартизировать и обеспечить встроенную реализацию операций клиентской части, такую как проверка данных. Тем не менее, серьезные динамические web-приложения до сих пор разрабатывать намного сложнее, чем традиционные настольные приложения, которые, предположительно, они должны заменить.

1.2. Что GWT делает для нас

Google Web Toolkit объединяет клиентский и серверный код в отдельном приложении, написанном на одном языке - Java. Это имеет ряд преимуществ. С одной стороны, довольно много разработчиков знают Java и JavaScript или Flash. С другой стороны, Java изобилит средствами разработки, такими как Eclipse, NetBeans и IDEA. GWT позволяет создавать web-приложения так же, как вы создаете Swing приложения, обеспечивая создание визуальных компонентов, установку обработчиков событий, отладку и т.д. - все в пределах стандартной IDE.

В пределах одного языка можно сделать общедоступным код на стороне клиента и сервера. Например, можно запустить один и тот же код для проверки данных - один раз на клиенте для быстрого ответа, и один раз на сервере для максимальной безопасности. Есть возможность даже перемещать код между уровнями при реорганизации приложения для соответствия измененным требованиям.

GWT абстрагирует DOM браузера, скрывая различия между браузерами для простой объектно-ориентированной разработки пользовательского интерфейса на основе шаблонов. Это способствует переносимости кода между всеми поддерживаемыми браузерами.

Если GWT уже кажется хорошим, то это лишь небольшая его часть. Необходимо быть осторожным при представлении некоторых отличий браузеров. Как сказал технический специалист Joel Spolsky, любая абстракция имеет свои дыры. Иногда необходимо углубляться в CSS/DOM/JavaScript для подавления причуд браузера в нестандартных программах. Но с GWT это скорее исключение, чем правило.

1.3. О книге

Эта книга предоставляет введение в Google Web Toolkit от установки, через создание первого приложения, до компонентов пользовательского интерфейса и удаленного вызова процедур, обучает входам и выходам библиотеки. Предполагаются некоторые

знания программирования на Java и HTML, но не обязательно быть экспертом web-программирования.

Хорошо, довольно разговоров - давайте начнем с первого GWT приложения!

Глава 2
Начало работы

Начало работы с Google Web Toolkit не сложное. В этой главе рассказывается, как установить необходимое программное обеспечение, а затем можно перейти непосредственно к созданию работающего приложения, используя GWT.

2.1. Поддерживаемые платформы

Разработка приложений GWT наилучшим образом поддерживается в Windows и Linux. Все примеры в этой книге сделаны под Windows.

Приложения GWT могут разворачиваться на web-сервере под любой операционной системой, и просматриваться любым современным браузером (IE6, IE7, Firefox, Opera и т. д.).

2.2. Установка

Перед началом разработки необходимо установить Java, среду разработки и сам GWT.

Java 1.4.2+

Сначала необходим дистрибутив Java. Так как GWT работает с Java 1.4.2 и выше, можно получить обновление Sun JDK 5.0 или 6.0 на сайте Sun. Для проверки вашей версии Java, запустите следующую команду из командной строки Windows:

```
C:\> java -version
java version "1.5.0_07"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_07-b03)
Java HotSpot(TM) Client VM (build 1.5.0_07-b03, mixed mode, sharing)
```

Eclipse

Затем необходим дистрибутив среды разработки Eclipse. Сейчас можно использовать и другие среды разработки Java, такие как NetBeans или IDEA, но разработчики Google, как и я, используют Eclipse. Поэтому она будет использоваться до конца этой книги.

Перейдите на страницу загрузки Eclipse, выберите версию 3.2 (или выше) и затем скачайте пакет Eclipse для вашей платформы (Windows, Linux, Mac и т. д.). Можно скачать либо полный SDK (включающий все исходные коды и документацию), или для меньшей загрузки лишь Platform Runtime Binary и JDT Runtime Binary.

Для более простого тестирования Eclipse, можно попробовать ее на сайте, который спонсирует Yoxos, или просто Eclipse от nexB.

GWT

Далее, необходимо загрузить Google Web Toolkit SDK (1.1.10 или выше). Разархивируйте Google Web Toolkit на вашу машину. Никакой специальной установки не требуется. Теперь вы готовы приступить к созданию первого проекта.

2.3. Создание сценариев запуска

Из командной строки, запустите эти команды (введите правильные пути для вашей системы):

```
C:\> mkdir c:\gwt-projects\MyProject
C:\> cd c:\gwt-projects\MyProject
C:\gwt-projects\MyProject> projectCreator -eclipse MyProject
Created directory C:\gwt-projects\MyProject\src
Created file C:\gwt-projects\MyProject\.project
Created file C:\gwt-projects\MyProject\.classpath
C:\gwt-projects\MyProject> applicationCreator -eclipse MyProject\
com.xyz.client.MyApp
Created directory C:\gwt-projects\MyProject\src\com\xyz
Created directory C:\gwt-projects\MyProject\src\com\xyz\client
Created directory C:\gwt-projects\MyProject\src\com\xyz\public
Created file C:\gwt-projects\MyProject\src\com\xyz\MyApp.gwt.xml
Created file C:\gwt-projects\MyProject\src\com\xyz\public\MyApp.html
Created file C:\gwt-projects\MyProject\src\com\xyz\client\MyApp.java
Created file C:\gwt-projects\MyProject\MyApp.launch
Created file C:\gwt-projects\MyProject\MyApp-shell.cmd
Created file C:\gwt-projects\MyProject\MyApp-compile.cmd
```

Команды `projectCreator` и `applicationCreator` - два shell-скрипта, являющиеся частью GWT, поэтому необходимо указать путь к ним или добавить директорию GWT в пути переменных окружения. `projectCreator` строит иерархию для общих GWT проектов, а `applicationCreator` добавляет простое GWT приложение, которое можно построить. `MyProject`, `MyApp` и `com.xyz` - названия примеров. Тем не менее, часть пакета .client является важной: к ней мы вернемся позже.

2.4 Запуск и отладка

Сейчас вы готовы к запуску приложения.

Запуск вне Eclipse

Во-первых, попробуем запустить приложение все IDE, используя один из предоставляемых shell-скриптов:

```
C:\gwt-projects\MyProject> MyApp-shell
```

Если все правильно работает, появится окно: `The GWT development shell` (это нечто подобное консоли) и окно браузера (рисунок 2.1).

Рисунок 2.1 - GWT приложение «Hello world»

Проверьте работоспособность приложения, кликнув по кнопке «`Click Me`» - должен появиться текст «`Hello World!`». Поздравляем, вы только что создали и запустили первое GWT приложение.

Запуск в Eclipse

Теперь закройте оба GWT окна, запустите Eclipse, и импортируйте этот проект в рабочую область (`File->Import->Existing Project into Workspace`). Проект будет построен, и, если все успешно, в результате вы получите нечто похожее на рисунок 2.2.

Затем выберите `Run->Debug` и нажмите на конфигурацию запуска `MyApp` (под Java Application). Затем нажмите на `Debug`. Два GWT окна должны появиться снова, так же как и на рисунке 2.1 на предыдущей странице.

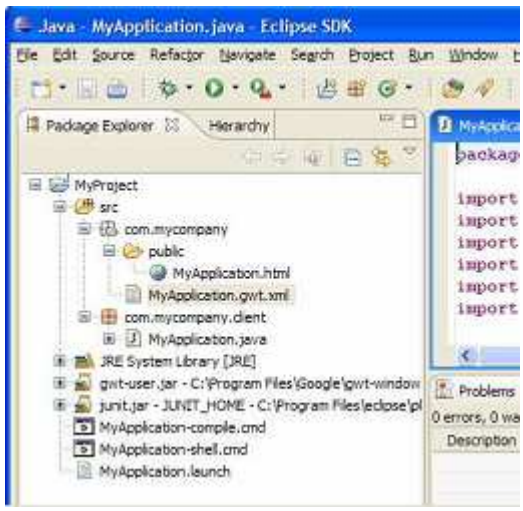

Рисунок 2.2 - Проект GWT в Eclipse

Отладка

Оставьте приложение запущенным и вернитесь в окно Eclipse. Установите точку останова на методе `onClick()` в `MyApp.java` двойным щелчком по пространству слева от строки (рисунок 2.3).

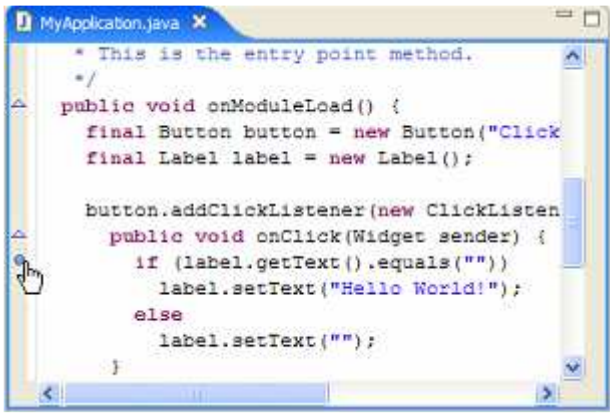

Рисунок 2.3 - Установка точки останова

Затем переключитесь в окно приложения и нажмите кнопку «`Click me`» снова. Eclipse остановится на Java коде там, где стоит точка останова. Можно сделать отдельный шаг, проверить переменные и т.д.

Итак, это Java код, а раньше можно было писать Ajax приложения, которые разворачивались в чистый JavaScript. Все преимущества вашей среды разработки - Eclipse, отладчик, реорганизация, редактор исходных кодов и т.д. - уже доступны в мире Ajax. Теперь вы видите потенциал этой технологии? В следующей главе мы взглянем за занавес, каким образом это сделано.

Глава 3
Hosted и Web режимы

В предыдущей главе, когда выполнялось GWT приложение, можно было заметить, что Google использует Hosted режим. Hosted режим используется только при разработке. Для отладки в реальных условиях приложение запускается в web режиме. Перед дальнейшей работой с GWT необходимо осознавать разницу между ними. Обратите внимание, что hosted режим доступен только для Windows и Linux.

3.1. Hosted режим

Hosted режим - тренировка GWT приложения. Это гибридная среда разработки, уникальная для GWT, позволяющая коду запускаться как настоящему коду Java, но в браузере. Выполнение в hosted режиме контролируется Google Web Toolkit.

Оболочка консоли разработки, по сути, является Eclipse Rich Client приложением, состоящим из консоли, сервера Tomcat и одного или нескольких hosted браузеров.

Hosted браузер имеет две связи со средой разработки. Первая - постоянное соединение http для получения web-страниц, файлов css, изображений и других ресурсов. Все это обрабатывается встроенным Tomcat сервером, используя сервлет `com.google.gwt.devshell.GWTShellServlet`.

Вторая - черный ход, который перехватывает все взаимодействия в hosted браузере и направляет их не в JavaScript, а в код Java в оболочке. Этот код Java вызывает реальный клиентский код, скомпилированный IDE в байт-код. Что именно происходит на этом этапе, скрыто в коде оболочки, распространяющейся с закрытым исходным кодом.

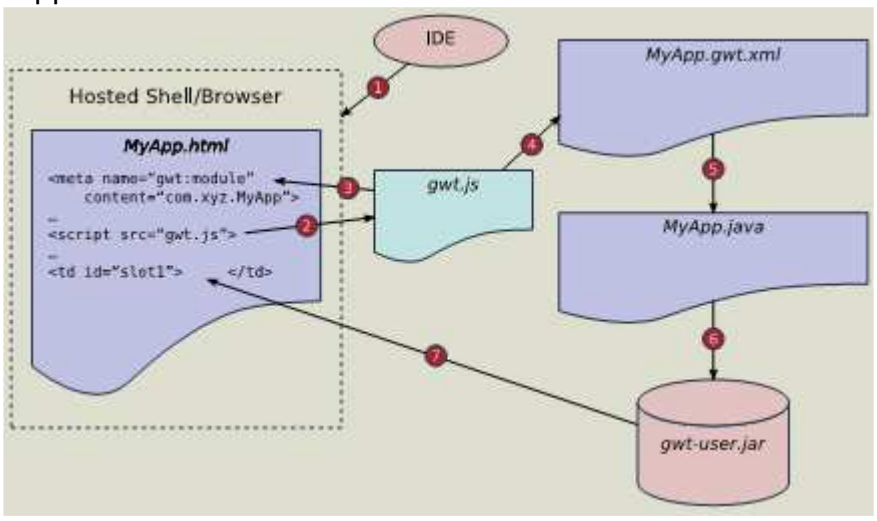

Рисунок 3.1 - Загрузка GWT страницы в Hosted браузер

На рисунке 3.1 изображена загрузка страницы в Hosted режиме:

1. Оболочка открывает окно Hosted браузера, которое загружает `MyApp.html`.
2. `MyApp.html` загружает `gwt.js` в теге `<script>`
3. `gwt.js` сканирует `MyApp.html` и просматривает `<meta name='gwt-module'>` в поисках заголовка модуля.
4. GWT читает файл модуля (`MyApp.gwt.xml`) в поисках точки входа (`MyApp`).
5. Создается объект класса `MyApp` и вызывается метод `onModuleLoad()`. Запускается приложение.
6. Код приложения делает вызов пользовательской библиотеки (`gwt-user.jar`), которая также является кодом Java.
7. Код `gwt-user.jar` управляет моделью DOM hosted браузера для добавления компонентов пользовательского интерфейса на web страницу и перенаправляет все события браузера назад в код приложения Java, используя специальные перехватчики в браузере.

Так как выполняется реальный код Java, можно использовать все утилиты, такие как отладчик Eclipse, findbugs, pmd, JUnit и т. д. Это почти так же, будто происходит разработка RIA с использованием Swing или SWT.

После отладки приложения и разработки тестов необходимо скомпилировать его в вид, пригодный для запуска в обычном браузере (без перехватчиков оболочки). Отсюда появляется web режим.

3.2. Web режим

При нажатии на кнопку `Compile/Browse` в hosted браузере, GWT переводит пакет `.client` в JavaScript и открывает обычный web браузер для просмотра приложения. При этом страницы все равно обслуживаются через встроенный Tomcat сервер, но они так же хорошо могут проходить и через файловую систему или обычный web сервер.

Другой путь запуска компилятора GWT заключается в использовании скрипта, поставляемого со сценариями запуска (`MyApp-compile`). Также по желанию для его выполнения можно написать скрипт `Ant`. Например, для управления сайтом `gwtpowered.org` есть скрипт `Ant`, который компилирует и копирует все на хостинг провайдера. Исходные коды можно найти на `http://code.google.coom/p/gwtpowered`.

Независимо от варианта запуска, компилятор GWT объединяет версию кода GWT API (похожий на `gwt-user.jar`) в одном файле JavaScript. Этот код и некоторые вспомогательные файлы

располагаются в директории www в проекте. Сюда же копируется все, что находится в директории public. Следующая таблица поясняет, что делает каждый из файлов:

Имя файла	Описание
длинное-шестнадцатиричное-имя.cache.html	Скомпилированный JavaScript
длинное-шестнадцатиричное-иимя.cache.xml	Определение выполнения
имя-модуля.nocache.html	Файл кэша
gwt.js	Общий загрузочный код GWT
history.html	Содержание истории iframe
MyApp.html	Главная страница, скопированная из public
tree*.gif	+/- картинки, используемые виджетом Tree

Последовательность выполняемых действий по мере загрузки страницы в web режиме (рисунок 3.2) немного отличается от загрузки в hosted режиме. Вот последовательность происходящего:

1. Web браузер загружает MyApp.html
2. MyApp.html загружает gwt.js в теге <script>
3. gwt.js сканирует MyApp.html и просматривает <meta name='gwt-module'> для получения названия модуля.
4. gwt.js модифицирует страницу для включения <iframe>, который способствует загрузке имя-модуля.nocache.html
5. Внутри имя-модуля.nocache.html JavaScript ищет поле браузера userAgent для определения того, какой браузер запущен пользователем (IE6, Mozilla, Opera и т. д.). Затем он выбирает подходящий код (файл кэша) для этого браузера и перенаправляет туда <iframe>.
6. Выполняется эквивалентный onModuleLoad() метод в JavaScript и приложение начинает работать с этого места. Взаимодействие с DOM браузера осуществляется с помощью обычных динамических вызовов HTML в скомпилированном JavaScript.

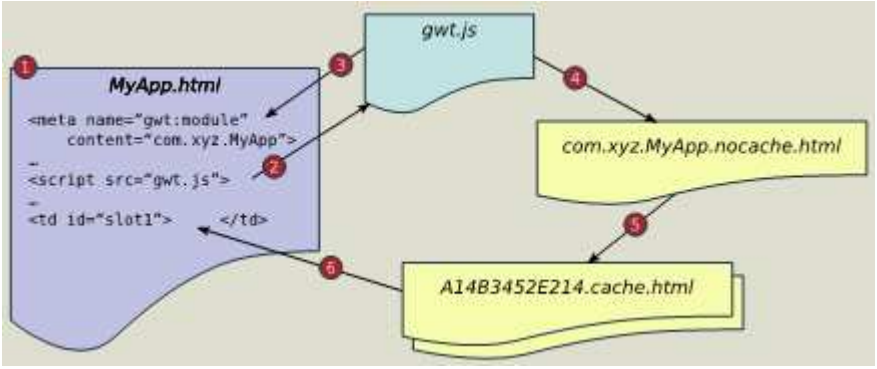

Рисунок 3.2 - Загрузка GWT страницы в web режиме

3.3. Искажение кода

По умолчанию, GWT компилятор Java в JavaScript создает искаженный выход. Искаженный код получается нечитаемым и сложно привести к исходному виду. С ним крайне сложно производить отладку. Если приходится производить отладку кода, полученного с помощью GWT, необходимо отключить искажение с помощью параметров командной строки GWT компилятора (например, передать аргументом в скрипт `MyApp-compile.cmd`). Опция `-style pretty` позволяет получить результат с читаемыми именами. Для того чтобы видеть полные типы Java как часть имен, необходимо использовать опцию `-style detailed`.

3.4. Развертывание

До сих пор все примеры зависели от hosted оболочки Tomcat сервера, который взаимодействовал со всеми файлами приложения. Тем не менее, в web режиме они могут быть портированы на любой web сервер или даже на локальную файловую систему (для тестирования). Чтобы испытать это, скопируйте содержимое папки `www` в другое место в файловой системе и направьте браузер на стартовую HTML страницу. Приложение будет работать абсолютно так же, как и раньше.

Для простых программ, таких как эта, не требуется взаимодействие с сервером, потому что нет серверного кода. Мы увидим некоторые усложненные программы в главе 5 (Удаленный Вызов Процедур), которые требуют больше, чем просто скопировать директорию, но на данный момент давайте посмотрим, какие преимущества мы имеем, используя GWT компоненты пользовательского интерфейса.

Вскоре вы заметите, что разработка GWT приложений подобна разработке настольных приложений с использованием Swing, SWT либо Visual Basic. Создаются элементы управления, такие как кнопки,

списки и таблицы, добавляются на родительские элементы, происходит взаимодействие с ними через слушателей событий. Вы располагаете их по определенным соглашениям и пытаетесь придать им наилучший вид с любым размером шрифта и разрешением экрана. Основным отличием является то, что GWT приложение будет запускаться в браузере, а значит должна быть HTML страница, которая его включает.

Обычно web приложения организованны как набор HTML страниц с некоторой навигацией. Например, имеется страница товаров, страница заказа и страница подтверждения. К сожалению, в GWT приложении приходится оставаться на одной странице все время. Вместо изменения web-страниц, изменяется содержание одной страницы для отображения текущего состояния. Например, могут быть различные панели для товаров, заказа и подтверждения заказа в пределах страницы, но в определенное время отображаться будет всего одна. Это предоставляет пользователю более гладкий и удобный интерфейс в сравнении с решением проблемы старым путем.

4.1. Связка с HTML

Если взглянуть в проект в `src/com/xyz/public`, можно обнаружить файл `MyApp.html`. Это область, в пределах которой будет расположен интерфейс пользователя GWT.

Каждое GWT приложение находится в отдельной HTML странице. Это может быть статическая страница, как эта, либо страница, сгенерированная серверным фреймворком, таким как JSP, Struts, Ruby on Rails и т. д. Для простоты мы будем рассматривать только статические страницы до конца книги.

Очевидно, что если файл `MyApp.html` находится в директории `public`, то он будет скопирован в результирующую версию для развертки на сервере. Если есть какие-либо изображения, таблицы стилей и т. д., их необходимо поместить в эту же директорию.

В заголовке HTML страницы необходимо указать обязательный тег meta, который свяжет эту страницу с GWT модулем.

```
<meta name='gwt:module' content='com.xyz.MyApp'>
```

Модуль GWT представляет собой код клиентского приложения и поддерживаемые вами ресурсы. Модуль `com.mycompany.MyApp` определяется в файле модуля `src/com/mycompany/MyApp.gwt.xml`.

```
<module>
<!-- Inherit the core Web Toolkit stuff. -->
```

```
<inherits name='com.google.gwt.user.User' />
<!-- Specify the app entry point class. -->
<entry-point class='com.xyz.client.MyApp' />
</module>
```

Здесь можно наблюдать имя класса Java. Когда загружена страница HTML, GWT просматривает тег meta, читает xml файл для получения имени класса и начинает выполнять код из класса `EntryPoint`. В GWT 1.1 также можно встроить `.css` файлы и другие ресурсы в заголовке модуля.

4.2. Точка входа

Ваш класс точки входа (`MyApp`) определяет интерфейс `EntryPoint` и реализует один метод `onModuleLoad()`. Этот метод отвечает за построение пользовательского интерфейса приложения GWT.

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

Код создает два элемента пользовательского интерфейса: `Button` и `Label`. Это примеры виджетов GWT, которые подобны другим виджетам библиотек Java, таких как Swing и SWT. Для просмотра поддерживаемых GWT виджетов смотрите часть 4.4. Виджеты.

Если посмотреть на файл HTML, внизу он ссылается на два контейнера динамического содержания:

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

Обратите внимание на данные этим ячейкам идентификаторы: `slot1` и `slot2`. В коде Java эти слоты относились к только что созданным `Button` и `Label`:

```
// 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);
```

RootPanel является лишь оболочкой HTML элементов на странице. Они создаются по мере необходимости. Приведенный код получает RootPanel для каждого из двух элементов <td>, на которые есть ссылка по идентификатору, а затем добавляет в них виджет GWT.

Эта таблица определяет, как виджеты располагаются на экране. Наиболее удобный способ определения расположения виджетов - использование панели. Панели GWT являются виджетами, которые могут содержать один или несколько виджетов, и организовать их заранее определенным способом. Например, можно создать HorizontalPanel, добавить на нее кнопку и текст, а затем добавить панель на RootPanel (как RootPanel.get().add(hPanel)). Список предопределенных панелей библиотеки GWT можно найти в разделе 4.5.

4.3. События

Web приложение будет довольно скучным, если мы не будем с ним взаимодействовать, так что давайте посмотрим на последний компонент - заставим кнопку что-то делать. При программировании с использованием Swing, мы добавляем слушатель кликов для JButton. В GWT необходимо делать то же самое.

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

Когда пользователь нажимает на кнопку, вызывается метод onClick(). В данном примере нет никакой логики, лишь меняется текст виджета Label либо на пустой, либо на «Hello World!». Приложение в действии изображено на рисунке 2.1.

4.4. Виджеты

Эта часть дает описание списка большинства встроенных виджетов GWT. Для каждого виджета приводится его назначение и основные методы. Для многих виджетам приведено изображение, чтобы понять, как они выглядят. Конечное отображение зависит от браузера пользователя и операционной системы. Информация о расположении виджетов на странице находится в разделе 4.5.

Цвета, шрифты и другая информация о стиле хранится в стандартных Каскадных Таблицах Стилей (CSS). Часто виджет предопределяет класс CSS (указан после описания), но можно добавить и свое имя класса виджету и создать ссылку на него в вашем `.css` файле.

`Button`

Стандартный виджет в виде кнопки. Для получения оповещений о нажатии его пользователем необходимо вызвать метод `addClickListener()`.

Стили CSS: `.gwt-Button { }`

`CheckBox`

Стандартный виджет в виде переключателя. Для получения оповещений о нажатии его пользователем необходимо вызвать метод `addClickListener()`. Для проверки, выбран ли переключатель, необходимо вызвать метод `isChecked()`.

Стили CSS: `.gwt-CheckBox {  }`

`Composite`

Виджет, который может быть обернут другим виджетом, скрывая методы обернутого виджета. Добавляется и выглядит на панели аналогично обернутому виджету.

`Composite` удобен для создания отдельных виджетов отдельно от других виджетов. Но так как класс Composite может обернуть лишь один виджет, часто приходится создавать панель, оборачивать ее Composite, а затем добавлять виджеты на обернутую панель.

`DialogBox`

Всплывающая форма, которая имеет область заголовка в верхней части и может перетаскиваться пользователем. Сделайте наследника класса `DialogBox` для вашего диалогового окна, вызовите в конструкторе `setText()` для установки заголовка и `setWidget()` для добавления содержимого в диалоговое окно (включая любую кнопку «`Закрыть`»). Вызывайте метод `show()` для открытия диалога и `hide()` для закрытия.

Стили CSS: `.gwt-DialogBox { }`
`gwt-DialogBox .Caption { }`

`FileUpload`

Виджет, который содержит элемент выбора файла. Может быть использован только внутри `FormPanel`. Метод `setName()` задает имя элемента, который будет вызывать подтверждение отправки на сервер.

`FlexTable` и `Grid`

Таблица, которая может содержать текст, разметку HTML или любой другой тип виджета. `FlexTable` создает ячейки и может быть растянута (каждый ряд может содержать разное число ячеек). `Grid` всегда содержит один фиксированный размер.
Для добавления ячеек используются методы `setText()`, `setHTML()` или `setWidget()`, для изменения формата ячеек используется метод `getFlexCellFormatter()`. Ячейки в `FlexTable` можно растягивать на несколько рядов или столбцов.

`Frame` и `NamedFrame`

Виджет, который оборачивает элемент HTML `<iframe>`, который может содержать другой web сайт. Обратите внимание, что если вы используете `History` (глава 6, История и закладки), любые записи истории из `Frame` будут упущены из истории приложения. Для установки адреса страницы используйте конструктор или метод `setUrl()`.

Стили CSS: `.get-Frame { }`

`HTML`

Виджет, содержащий HTML разметку. Если необходим просто текст, лучше использовать виджет `Label`. Вставка HTML разметки, особенно содержащей данные, введенные пользователем, может привести к потенциальным угрозам безопасности с использованием межсайтового кода при неосторожности.

Стили CSS: `.gwt-HTML { }`

`Image`

Виджет, отображающий картинку по заданному URL. Для установки адреса изображения необходимо вызвать конструктор или метод `setUrl()`. С помощью метода `addLoadListener()` также можно обработать событие полной загрузки изображения либо возникшей ошибки при загрузке.

Стили CSS: `.gwt-Image { }`

`Hyperlink`

Виджет, который содержит гиперссылку либо ссылку на другое состояние запущенного приложения. По клику на нее, создается запись в истории с помощью `History.newItem()`, но без перезагрузки страницы. Для создания ссылки на другую страницу (например, на другой сайт), необходимо использовать виджет HTML.

Стили CSS: `.gwt-Hyperlink { }`

`Label`

Виджет, содержащий текст, не интерпретируемый как HTML. Поддерживает перенос слов и установку горизонтального выравнивания. Для изменения текста используется метод `setText()`.

Стили CSS: `.get-Label { }`

`ListBox`

Виджет, представляющий список выбора для пользователя в виде простого или выпадающего списка. Записи добавляются с помощью метода `addItem()`, высота списка устанавливается методом `setVisibleCount()`. Если сделать высоту равной 1, список превращается в выпадающий список. Для взаимодействия с выделенными элементами используются методы `getSelectedIndex()` или `isItemSelected()`.

Стили CSS: `.gwt-ListBox { }`

`MenuBar` и `MenuItem`

Стандартный виджет меню. `MenuBar` может содержать любое число записей меню, каждая из которых может либо побуждать

команду, либо открывать каскадное меню. Для добавления пунктов меню используется метод addItem().

Стили CSS: .gwt-MenuBar { само меню }
.gwt-MenuItem { пункт меню }
.gwt-MenuItem-selected { выделенный пункт меню }

PasswordTextBox

Подобен обычному тексту для ввода, кроме того, что ввод визуально скрывается браузером для обеспечения исключения подглядывания.

Стили CSS: .gwt-PasswordTextBox { }

RadioButton

Виджет, представляющий радио кнопку, управляемую вручную. Для получения события о нажатии кнопки используйте метод addClickListener(), для проверки, выбрана кнопка или нет, используйте метод isChecked().

TabBar

Горизонтальная полоса вкладок, подобных папкам, обычно используется как часть TabPanel. Для добавления элементов используйте метод addTab(), для получения уведомлений перед выбором вкладки используйте метод onBeforeTabSelected(), после - onTabSelected().

Стили CSS: .gwt-TabBar { сама полоса вкладок }
.gwt-TabBarFirst { пространство справа от полосы }
.gwt-TabBarRest { пространство справа от полосы }
.gwt-TabBarItem { вкладки }
.TabBarItem-selected { добавочный стиль для выбранных вкладок }

TextArea

Поле текста, поддерживающее многострочный ввод. Методы setCharacterWidth() и setVisibleLines() устанавливают

размеры поля ввода. Методы getCursorPos(), getSelectionLength() и getSelectedText() дают информацию о текущем выделенном тексте. Также можно программно установить текущий выделенный текст. Для получения сообщений о нажатии клавиш используется метод addkeyboardListener().

Стили CSS: .gwt-TextArea { }

TextBox

Текстовое поле для однострочного ввода. Для установки размера поля используется метод setVisibleLength(). Для получения информации о текущем выделенном тексте используются методы getSelectionLength() и getSelectedText(). Также можно программно установить текущий выделенный текст. Для получения сообщений о нажатии клавиш используется метод addkeyboardListener().

Tree и TreeItem

Виджет, представляющий стандартное иерархическое дерево. Tree содержит иерархию объектов TreeItem, которые пользователь может открывать, закрывать и выбирать. Для добавления записей в корень дерева используйте метод addItem() для объекта класса Tree. Записи могут быть как строками HTML, так и объектами TreeItem, содержащими поддеревья. Для добавления поддерева вызовите метод addItem() для объекта класса TreeItem.

Стили CSS: .gwt-Tree { само дерево }
.gwt-Tree-Item { запись дерева }
.gwt-TreeItem-selected { выбранная запись дерева }

4.5 Панели

Панель - виджет, содержащий другие виджеты (а также другие панели). Используется для расположения виджетов в таблицах, плоскостях, рядах или колонках. Этот раздел дает описание встроенным в GWT панелям. Также смотрите раздел 4.4 Виджеты.

AbsolutePanel

Эта панель располагает все дочерние компоненты в абсолютных координатах (в CSS: position: absolute), позволяя им

перекрываться. Виджеты располагаются в координатной системе документа. Обратите внимание, что эта панель не будет автоматически изменять размеры для обеспечения достаточного пространства дочерним элементам, расположенным в абсолютных координатах. Она должна быть определенно масштабирована для обеспечения такого пространства.

`DeckPanel`

Панель, отображающая все дочерние виджеты на «плоскости», на которой лишь один может быть виден в данный момент. Все дочерние элементы распределяются в памяти браузера таким образом, чтобы хватало памяти для реализации больших панелей.

Метод `add()` добавляет виджет на плоскость, метод `showWidget()` делает один виджет видимым. `DeckPanel` используется как тело `TabPanel`.

`DockPanel`

Панель, располагающая дочерние элементы пристыкованными к их внешним краям, а центральный виджет занимает оставшееся пространство. Можно установить горизонтальное и вертикальное выравнивание отдельных ячеек.

Обратите внимание, что поведение этой панели может быть разное в зависимости от обстоятельств. Например, если сначала добавлен элемент с параметром `WEST`, он займет верхний левый угол, а если первым добавлен элемент с параметром `NORTH`, то верхний левый угол займет он.

`FlowPanel`

Панель, форматирующая дочерние виджеты, используя по умолчанию поведение формата HTML. Используйте метод `add()` для добавления нового дочернего виджета на панель. Для горизонтального размещения элементов на панели используйте CSS стиль `display: inline`.

`FocusPanel`

Простая панель, позволяющая устанавливать фокус на ее содержимое, а так же имеет возможность обрабатывать события мыши и клавиатуры. Используйте конструктор или метод `setWidget()` для установки виджета на панели.

`FormPanel`

Панель, которая оборачивает содержимое в тег HTML `<form>`. `FormPanel` может содержать только `TextBox`, `PasswordTextBox`, `RadioButton`, `CheckBox`, `TextArea`, `ListBox`, `FileUpload`. Используйте метод `setName()` для дочерних элементов, чтобы связать их с полями формы, которая отправляется на сервер. Для установки URL адреса, по которому будет отправляться форма, используйте метод `setAction()`, а метод `submit()` для отправки формы. Вызовите `addFormHandler()` для получения уведомления о начале и завершении отправки формы.

`HorizontalPanel`

Панель, которая распо0лагает все виджеты в один ряд. Используйте метод `add()` для добавления виджета на панели. Также можно установить горизонтальное и вертикальное выравнивание отдельных ячеек.

Совет: устанавливайте свойство выравнивания перед добавлением дочерних элементов. При таком подходе не придется устанавливать его для каждого дочернего элемента.

`HTMLPanel`

Панель, содержащая HTML с заменяемыми частями. Используйте конструктор для установки начальной строки HTML. Внутри этой строки некоторые элементы должны содержать `id=атрибут`, позволяя в дальнейшем использовать метод `add()` для добавления дочернего виджета элементу с идентификатором. При создании строки HTML следует использовать метод `createUnique()`, потому что любая пара элементов на странице не должна содержать одинаковые идентификаторы.

`PopupPanel`

Всплывающая панель может «всплывать» над виджетом. Она перекрывает клиентскую область (и любые предыдущие всплывающие окна). Чтобы использовать всплывающую панель, создайте класс-наследник от `PopupPanel`, и вызовите метод `setWidget()` в конструкторе. Метод `show()` отображет панель, метод `hide()` убирает ее. Если необходимо просмотреть (и, возможно, обработать) события клавиатуры перед тем, как они попадут в любой другой виджет, перегрузите методы

onKeyDownPreview(), onKeyPressPreview() или ponKeyUpPreview().

ScrollPanel

Простая панель, помещающая содержимое в прокручиваемую область. Используйте конструктор или метод setWidget() для помещения виджета в содержимое панели.

StackPanel

Панель, располагающая дочерние элементы вертикально в стек, отображая каждый раз только один из них, причем остальные дочерние элементы отображаются в виде заголовков, на которые может нажать пользователь.

Стили CSS:

```
.gwt-StackPanel { сама панель }
.gwt-StackPanelItem { невыбранные записи }
.gwt-StackpanelItem-selected{выбранные записи }
```

TabPanel

Объединение TabBar и DeckPanel, которое показывает набор виджетов по вкладкам. Лишь один виджет видим в определенное время, в зависимости от выбранной вкладки. Чтобы использовать панель вкладок, создайте новый класс, расширяющий TabPanel и вызовите метод add() для добавления каждого виджета на панель. Реализуйте метод onBeforeTabSelected() при необходимости просмотра и, возможно, обработки событий выбора вкладки, или метод onTabSelected() для получения уведомления после происхождения этого события.

VerticalPanel

Панель, располагающая все свои виджеты в одну вертикальную колонку. Для добавления виджетов на панель используется метод add(). Можно также установить горизонтальное и вертикальное выравнивание отдельных ячеек.

Совет: устанавливайте свойство выравнивания перед добавлением дочерних элементов. При таком подходе не придется устанавливать его для каждого дочернего элемента.

Эта глава раскрыла основы, которые необходимы для создания пользовательского интерфейса на GWT. В следующей главе разбирается другая мощная черта GWT, позволяющая клиентскому и серверному коду общаться друг с другом - удаленный вызов процедур.

Глава 5
Удаленный вызов процедур

Долгие годы разработчики бились в поисках идеального расположения исполняемого кода. Следует ли его запускать на сервере, где он будет иметь централизованный контроль и будет защищен, либо же следует его запускать ближе к пользователю на его компьютере для получения преимуществ локальной вычислительной мощности и взаимодействия? GWT позволяет части приложения запускаться и там и там с помощью взаимодействия между ними, с использованием удаленного вызова процедур (RPC).

5.1. Где живет ваш код?

Толстые клиенты, то есть стандартные настольные приложения, полностью запускаются на пользовательской машине. Например, это Microsoft Word. Приложение устанавливается на машину пользователя, и все его файлы также хранятся здесь.

Тонкие клиенты работают несколько иначе. Они запускаются на некоторой общей машине, а настольный компьютер используется только для пользовательского интерфейса. Большое число web-приложений входят в эту категорию, например, Ebay и Amazon. На настольный компьютер ничего не устанавливается, кроме стандартного средства просмотра (браузера).

Существует множество других технологий, занимающих промежуточную область (богатые, умные, клиент/серверные и т.д.). Приложения Ajax в общем, и, в частности, GWT, попадают в класс Богатых приложений Интернет (RIA). Часть кода запускается на клиентской машине, часть - на серверной.

Подобно тонкому клиенту, RIA не требуют установки никаких компонентов на компьютер, кроме стандартного браузера. Но подобно толстым клиентам, пользовательский интерфейс может быть достаточно богат и интерактивен, получая некоторые преимущества настольного компьютера.

5.2. Вызов удаленного кода

Каждый раз, когда необходимо запускать код в двух разных местах, необходимо налаживать связь между ними. Наиболее простым способом для этого является удаленный вызов процедур.

В данном случае, удаленный вызов процедур есть лишь способ, с помощью которого клиент может выполнять некоторую логику на сервере и получать результаты. Например, нам необходимо получить текущее положение или найти координаты на карте заданного адреса. RMI, .NET Remoting, SOAP, REST и XML-RPC являются все протоколы

удаленного вызова процедур с которыми вы скорее всего знакомы. GWT не использует ни один из них.

5.3. Почему новый протокол?

Первой причиной, по которой GWT использует отдельный протокол, является то, что вызовы из браузера асинхронны. Это A в Ajax, не так ли? Мы делаем запрос, а он должен (или не должен) вернуть нам ответ когда-то в будущем. В то время, пока запрос обрабатывается, GWT приложение должно заботиться о своих делах и привыкать к показу того, что данных еще нет, как можно лучше.

Например, если нажать на сообщения в GMail, клиент запросит текст сообщения с сервера и напишет строку `Loading…` в углу окна браузера. Если надоело ждать ответа сервера, можно либо нажать кнопку «`Назад`», либо выбрать другое сообщение, либо нажать на любую другую вкладку браузера. Обратите внимание, что пользовательский интерфейс не блокируется, как было бы при синхронном вызове. Пока GMail не написан на GWT (как сейчас), идея подобная.

Во-вторых, GWT RPC должен быть простым. Браузер загружает весь клиентский код, как только пользователь начинает работу с приложением, и, если он более 100Кб, пользователь заметит паузу. Между прочим, JavaScript выполняется в браузере также довольно медленно. Так что при реализации сложного протокола, как SOAP, возможно получилось бы слишком много кода для загрузки и слишком низкая скорость для работы.

Наконец, JavaScript не поддерживает Java-подобную сериализацию и загрузку динамических классов, так что GWT RPC не может пользоваться этим. С тех пор, как ни один из существующих протоколов не смог удовлетворить всем этим требованиям, Google предложил свой. К счастью, реализация протокола с открытым кодом, так что вы можете легко его приручить для удовлетворения каких-либо потребностей, если понадобится.

GWT RPC не документирован. Тем не менее, его можно исследовать по исходным кодам, но этого делать не рекомендуется, так как он может быть изменен без предупреждения. GWT разработан так, что клиентская и серверная часть разворачиваются одновременно, как две части одного приложения. Если требуется протокол, который не будет изменяться при обновлениях, GWT RPC лучше не использовать.

5.4. Основы GWT RPC

GWT основан на Java: Java на клиентской стороне (преобразованная в JavaScript для развертки в браузере) и Java на

стороне сервера (запускается как обычный сервлет Java). Тем не менее, можно использовать другой язык на серверной стороне (такой как PHP, Python или Ruby), но это будет больше борьба за принципы. Так что лучше делайте как все и просто используйте Java.

Первым шагом для создания удаленного вызова процедуры является определение шаблона, описывающего вызов. К сожалению, вам придется делать это 3 раза: один раз на сервере, один раз на клиенте и один раз в интерфейсе, который они разделяют. GWT использует стандартную договоренность об именовании для их объединения.

Предположим, на клиенте необходимо проверить цены в стоковых магазинах. Для обеспечения чатовой взаимосвязи, необходима возможность сразу опросить несколько магазинов. Необходим метод, который принимает массив имен и возвращает массив double. Создадим этот метод в интерфейсе `StockService`. Необходимо определить три интерфейса:

Имя	Расположение	Цель
интерфейс StockService	Клиент и сервер	Описывает сервис (его сигнатура)
класс StockServiceimpl	Сервер	Здесь находится сам код
интерфейс StockServiceAsync	клиент	Позволяет вызвать сервис

Единственным объектом, имеющим здесь реализацию, является класс `StockServicImpl`, так что рассмотрим его первым. Если разработка проходит в Eclipse, следует создать проект `RpcExample` обычным способом (либо скачать примеры для этой книги). Затем добавить класс в пакет `com.xyz.server`, так как он должен находиться на сервере:

```
public class StockServiceImpl extends RemoteServiceServlet implements
      StockService {
      public double[] getPrices(String[] symbols) {
            double[] result = new double[symbols.length];
            for (int i = 0; i < symbols.length; i++) {
                  result[i] = getPrice(symbols[i]);
            }
            return result;
      }
}
```

Предположим, что все товары стоят $400 и их цена возрастает на $1 каждый раз, когда мы проверяем рынок (хотелось бы купить несколько вещей пока они по $4). Это поможет продемонстрировать следующий код:

```
private double price = 400.0;
private synchronized double getPrice(String symbol) {
      // Real code would retrieve the prices here
      return price++;
}
```

Возвращаясь на сторону клиента, создаем класс `EntryPoint` для тестовой программы. Он будет выглядеть предельно знакомым сейчас, исключая небольшие изменения в реализации интерфейса `ClickListener` и изменении типа виджета `Label`:

```
public class RpcExample implements EntryPoint, ClickListener {
	private Button button = new Button("Click me");
	private HTML label = new HTML();
	public void onModuleLoad() {
		button.addClickListener(this);
		RootPanel.get("slot1").add(button);
		RootPanel.get("slot2").add(label);
	}
	// ...
}
```

По нажатию кнопки вызывается метод `onClick()`. Здесь происходит вызов метода с сервера:

```
private String[] symbols = { "GOOG", "MSFT", "SUNW" };
public void onClick(Widget sender) {
	// (1) Create the client proxy. Note that although you are
	// creating the service interface proper, you cast the
	// result to the async version of the interface. The cast
	// is always safe because the generated proxy implements
	// the async interface automatically.
	StockServiceAsync service = (StockServiceAsync) GWT
		.create(StockService.class);
	// (2) Specify the URL at which our service implementation is
	// running. Note that the target URL must reside on the same
	// domain and port from which the host page was served.
	ServiceDefTarget endpoint = (ServiceDefTarget) service;
	endpoint.setServiceEntryPoint(GWT.getModuleBaseURL() + "prices");
	// (3) Create an async callback to handle the result.
	AsyncCallback callback = new AsyncCallback() {
		public void onSuccess(Object result) {
			double[] prices = (double[]) result;
			updatePrices(symbols, prices);
		}
		public void onFailure(Throwable caught) {
			// do some UI stuff to show failure
		}
	};
	// (4) Make the call. Control flow will continue immediately
	// and later 'callback' will be invoked when the RPC completes.
	service.getPrices(symbols, callback);
	}
}
```

Обратите внимание на `StockServiceAsync` и `StockService` на первом шаге. Эти интерфейсы могут быть созданы вручную или с помощью какой-либо утилиты (например, скрипта на Ruby) из класса StockServiceImpl следующим образом:

```
public interface StockServiceAsync {
	void getPrices(String[] symbols, AsyncCallback callback);
}
```

```
public interface StockService extends RemoteService {
      double[] getPrices(String[] symbols);
}
```

Теперь взглянем на шаг 3 в методе `onClick()`. Когда завершается метод `getPrices()` (предполагаем, что он завершается), выполняется метод `onSuccess()` из `AsyncCallback`. Он в свою очередь вызывает `updatePrices()`, который изменяет пользовательский интерфейс, отображая результаты. Определение этого метода следующее:

```
private void updatePrices(String[] symbols, double[] prices) {
      String html = "";
      for (int i = 0; i < symbols.length; i++) {
            html += symbols[i] + ": " + prices[i] + "<br/>";
      }
      label.setHTML(html);
}
```

При запуске этой программы и нажатии на кнопку ничего не произойдет. Hosted режим укажет на ошибку:

```
[TRACE] The development shell servlet received a request for
'prices' in module 'com.xyz.RpcExample'
[WARN] Resource not found: prices
```

Это потому, что пока мы не развернули серверный код. В Web режиме необходимо развернуть сервлет `StockServiceImpl` в контейнере, например, в Tomcat. Используйте документацию вашего контейнера сервлета для получения информации как это делается. В hosted режиме все гораздо проще. Все, что требуется – добавить определение модуля (`RpcExample.gwt.xml`):

```
<!-- Specify the servlet class. -->
<servlet path="/prices" class="com.xyz.server.StockServiceImpl" />
```

Теперь необходимо перезапустить приложение (Refresh в hosted браузере) и можно получить следующее:

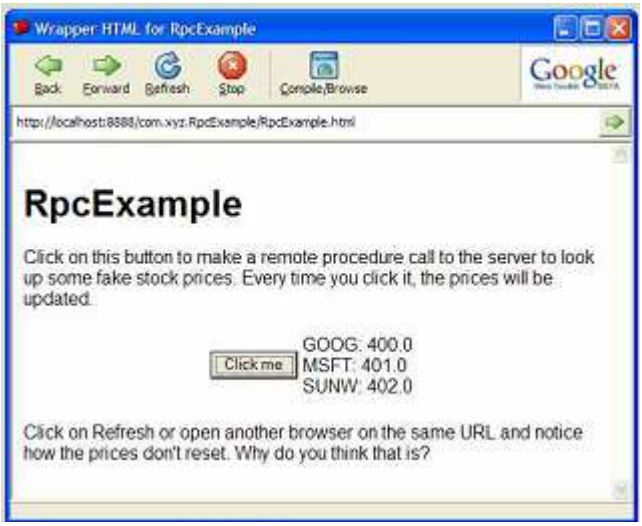

Рисунок 5.1 – Пример использования RPC для стоковых магазинов

Состояние сервера

При каждом нажатии на кнопку, цены возрастают, даже если делать это в разных браузерах. Это происходит потому, что экземпляр сервлета запущен для всех клиентов. Для того, чтобы каждый пользователь взаимодействовал с разным экземпляром сервлета, необходима реализация некоторого идентификатора сессии, возможно какого-либо значения, которое проходит все этапы RPC.

5.5. Сериализация

В предыдущей главе мы отправляли несколько строк и получали несколько значений double в ответ. При реализации этого механизма данные проходят сериализацию, то есть преобразуются из объектов Java в байты, и обратно в объекты Java на другой стороне. Так же, как и с использованием RMI и .NET Remoting, нет ограничений на использование лишь примитивных типов данных. Любой сериализующийся тип может быть передан параметром или возвращен значением из удаленного вызова. Обратите внимание, что идея сериализации GWT отличается от идеи сериализации Java. Тип есть «GWT сериализующимся», если он:

- примитивный, такой как `char`, `byte`, `short`, `int`, `long`, `boolean`, `float` или `double`;
- оболочка примитивного типа (`Character`, `Byte` и т. д.);
- `String` или `Date`;
- массив сериализуемых типов (включая массив массивов);
- пользовательский класс, который содержит только сериализующиеся типы;

- реализует интерфейс `IsSerializable`.

Большинство простых типов и классов, которые приходится использовать, сериализуются автоматически. Например:

```
public class Rect implements IsSerializable {
      private int height;
      private int width;
      // Must have a zero-arg constructor, or no constructor
      public Rect() {
      }
      public void setHeight(int height) {
            this.height = height;
      }
      // ...
}
```

Классы-коллекции, такие как `Set`, `List`, `Map` и `HashMap`, являются ограниченными: необходимо использовать специальную аннотацию в JavaDoc для указания компилятору GWT, какие объекты будут в коллекции. Например:

```
public class MyClass implements IsSerializable {
      /**
      * This field is a Set that must always contain Strings.
      *
      * @gwt.typeArgs <java.lang.String>
      */
      public Set setOfStrings;
}
```

Обычно, это делается для аннотации параметров и возвращаемого значения:

```
public interface MyService extends RemoteService {
      /**
      * The first annotation indicates that the parameter named 'c' is
      * a List that will only contain Integer objects. The second
      * annotation indicates that the returned List will only contain
      * String objects (notice there is no need for a name, since it
      * is a return value).
      *
      * @gwt.typeArgs c <java.lang.Integer>
      * @gwt.typeArgs <java.lang.String>
      */
      List reverseListAndConvertToStrings(List c);
}
```

В следующей главе будет рассматриваться проблема обработки кнопки «`Назад`».

Глава 6
История и закладки

Начиная с появления первой версии NCSA Mosaic web-браузеры были оснащены кнопкой «Назад». Эта функция была чрезвычайно простой при работе со статическими HTML страницами, но с появлением динамических web-приложений она стала головной болью.

Закладки (часто называемые Избранное) также является частичной проблемой для Ajax приложений, потому что пользователь взаимодействует с отдельной страницей, у которой различные секции или состояния находятся на одном и том же URL адресе.
К счастью, GWT решил обе эти проблемы.

6.1. Запись истории

Секретом обработки кнопки «Назад» и разрешения пользователю сохранять полезные закладки является запись истории. Запись истории – простая строка, которую необходимо сохранять для восстановления определенного состояния. Например, можно использовать ее для сохранения заголовка текущей вкладки на странице с несколькими вкладками, либо можно зашифровать в ней некоторое сложное состояние.

Google не документирует максимальную длину записи истории, но мы рекомендуем делать ее короткой, скажем, до 100 символов.

Текущая запись истории меняется при нажатии пользователем на гиперссылку либо кнопку «Вперед» или «Назад» в браузере.Также можно программно менять запись истории вызовом методов `History.newItem()`, `History.back()` или `History.forward()`.

6.2. Слушатель истории

Итак, как вы думаете, как работает запись истории? Конечно же, посредством регистрации слушателя событий. Определяется класс, который реализует `HistoryListener` и его метод `onHistoryChanged()`, затем проходит через `History.addhistoryListener()`. Многие просто используют класс `EntryPoint` для этих целей. Все слушатели удаляются при выходе из приложения.
При первом запуске приложения (при вызове метода `onModuleLoad()`), слушатель истории не вызывается. Это дает вам возможность определить, есть ли начальная запись или нет, выполнить какую-либо специальную инициализацию, а затем вызвать `onHistoryChanged()` вручную.

6.3. Как это работает

Пользователь видит, что запись истории появляется в адресной строке браузера как часть URL, например, http://www.gwtpowered.org/#Xyzzy. Если пользователь переходит по этому адресу, приложение GWT загружается и может вызвать `History.getTocken()` для обнаружения предполагаемых действий.

За реализацию этого механизма отвечает специальный тег `<iframe>`, который помещается на HTML страницу. GWT это делает автоматически:

```
<!-- OPTIONAL: include this if you want history support -->
<iframe id="__gwt_historyFrame"
style="width:0;height:0;border:0"></iframe>
```

Кнопки браузера «`Назад`» и «`Вперед`» просто хранят стек адресов, которые вы посетили. При осторожном управлении этим стеком, GWT относит их к записям истории, которые установило приложение.

Как вы, наверное, уже догадались, закладки следуют из этого механизма, потому что они хранят полный адрес (включая все после #). Также можно публиковать адреса в статьях или книгах, как сделано здесь.

6.4. Пример

Этот пример показывает, как обрабатывать начальные записи, устанавливать слушатели событий и обрабатывать события.

```
public class HistoryExample implements EntryPoint, HistoryListener {
      private Label lbl = new Label();
      public void onModuleLoad() {
            // Create three hyperlinks that change the application's
            // history.
            Hyperlink link0 = new Hyperlink("link to foo", "foo");
            Hyperlink link1 = new Hyperlink("link to bar", "bar");
            Hyperlink link2 = new Hyperlink("link to baz", "baz");

            // If the application starts with no history token, start it
            // off in the 'baz' state.
            String initToken = History.getToken();
            if (initToken.length() == 0)
                  initToken = "baz";

            // onHistoryChanged() is not called when the application
            first
            // runs. Call it now in order to reflect the initial state.
            onHistoryChanged(initToken);
            // Add widgets to the root panel.
            Panel panel = new VerticalPanel();
```

```
            panel.add(lbl);
            panel.add(link0);
            panel.add(link1);
            panel.add(link2);
            RootPanel.get().add(panel);

            // Add history listener
            History.addHistoryListener(this);
        }

        public void onHistoryChanged(String historyToken) {
            // This method is called whenever the application's history
            // changes. Set the label to reflect the current token.
            lbl.setText("The current history token is: " +
            historyToken);
        }
    }
```

При запуске этого кода отобразится запись с тремя гиперссылками, которые позволят установить запись (рис. 6.1). Нажмите на ссылки и на кнопки «Назад» и «Вперед», чтобы убедиться, что все работает как и ожидалось.

Рисунок 6.1 – Пример работы с историей

Как и большинство возможностей Ajax, GWT делает управление историей простой, скрывая весь необходимый код JavaScript, который необходим для реализации. Но каждый программист знает, что иногда необходимо марать руки. Следующая глава покажет, как это делать.

Глава 7
JavaScript native интерфейс

Иногда при написании Java кода (особенно при системном программировании) мы обнаруживаем, что приближаемся к железу и выполняем код вне виртуальной Java-машины. Например, может понадобиться доступ к библиотеке на другом языке. Для этого в Java метод определяется как native и обеспечивает реализацию этого метода на другом языке, таком как C.Это называется Java native интерфейс (JNI).

То же самое можно делать и с кодом Java в GWT на клиенте, только вместо кода на C native языком для браузера будет JavaScript. Таким образом, Google представляет JavaScript Native интерфейс (JSNI).

7.1. Определение native-метода

Для определения native-метода в JSNI используется стандартное ключевое слово Java native аналогично JNI. В JNI код на C определяется в отдельном файле, компилируется отдельно и динамически загружается во время выполнения. В JSNI native код JavaScript включается прямо в исходный код Java в специальном комментарии:

```
public class Alert {
      public static native void alert(String msg) /*-{
      $wnd.alert(msg);
      }-*/;
}
```

Блок JSNI комментария начинается с `/*-{`и заканчивается `}-*/`. Как показывает этот пример, при доступе к окну браузера и объекту документа из JSNI используются ссылки на `$wnd` и `$doc` соответственно. Скомпилированный скрипт запускается во фрейме и `$wnd` и `$doc` автоматически инициализируются для обеспечения правильной ссылки на окно с родительской страницей и документ во фрейме.

7.2. Как это работает

Как описано в разделе 3.2, компилятор GWT преобразует клиентскую часть Java программы в JavaScript. Когда компилятор видит определение метода, код, находящийся между скобок, проходит некоторый процесс трансляции. Если же это native метод, работа компилятора упрощается. Все что он должен сделать – скопировать native код прямо в результаты компиляции.

Если вы используете компилятор Microsoft Visual C++ или GNU C++, эффект будет подобен встроенному ассемблерному коду, только на более высоком уровне языка, чем ассемблер.
С тех пор, как JavaScript стал интерпретируемым, любые ошибки в коде JavaScript не будут отображаться до запуска.

7.3. Вызов JSNI из Java

Вызов JSNI метода из Java не отличается от вызова обычного метода Java. Например:

```
button1.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
            Alert.alert("clicked!");
      }
});
```

Вызывающий метод не может определить, вызываемый метод native или нет. Это дает некоторую гибкость в последующей смене взглядов на то, как реализуется метод.

7.4. Вызов Java из JSNI

Вызов в другую сторону немного сложнее. Предположим, мы передали объект в метод JSNI и необходимо получить его поле или вызвать его метод. Необходимо узнать, каким образом компилятор GWT преобразует поля Java и имена методов, чтобы получить к ним доступ из JavaScript.

Получение доступа к полям Java

Сторонники объектно-ориентированного подхода утверждают, что не следует предоставлять доступ к полям класса java напрямую, потому что это затрудняет преобразование кода в дальнейшем. Но здесь мы пишем native код, поэтому некоторыми правилами можно пренебречь. Синтаксис доступа к полям Java класса следующий:

```
Obj.@class: field
```

где
`obj` – экземпляр объекта, на который мы ссылаемся. Для переменных класса описание экземпляра необходимо опустить.
`class` – полный путь к классу, в котором определено поле (либо его подкласс).
`field` – имя поля, к которому необходимо получить доступ.

Выполнение методов Java

При вызове методов использует синтаксис, аналогичный доступу к полям. Отличие состоит в том, что необходимо также указывать сигнатуру вызываемого метода. Причина этого заключается в том, что методы Java могут быть перегруженными, то есть два метода могут иметь одинаковые имена и принимать разные параметры. Синтаксис следующий:

```
obj.@class::method(stg)(args)
```

где
`obj` – экземпляр объекта, для которого вызывается метод. Для методов класса его необходимо опустить.
`class` – полный путь к классу, в котором определен метод (либо его подклассу).
`method` – заголовок вызываемого метода.
`stg` – сигнатура внутреннего метода Java (см. раздел 7.4, Сигнатуры методов).
`args` – список фактических параметров, передаваемых в метод.

Сигнатура метода

Сигнатура методов JSNI такая же, как и сигнатура методов JNI, отличие лишь в том, что опускается возвращаемый тип. Тип опускается потому, что не требуется указывать, какой именно перегруженный метод вызывается. Следующая таблица показывает сигнатуру типов:

Сигнатура типа	Тип Java
Z	`boolean`
B	`byte`
C	`char`
S	`short`
I	`int`
J	`long`
F	`float`
D	`double`
Lполностью_определенный_класс;[тип	`полностью_определенный_класс тип()`

Например, следующий метод Java:

```
long f (int n, String s, int[] arr);
```

будет иметь следующую сигнатуру:
`(ILjava//lang/String;[I)`

На рисунке 7.1 изображены правила прохождения переменных через JSNI и их возвращения:

Тип java	Представление в коде JavaScript
число Java	число JavaScript, например, `var x = 42;`
`String`	строка JavaScript, например, `var s = “моя строка”;`
`Boolean`	логическое значение JavaScript, например, `var b = true;`
`JavaScriptObject`	объект JavaScript, который должен быть определен в JavaScript коде, обычно как возвращаемое значение некоторого JSNI метода
Массив Java	значение, которое может только передаваться назад в код Java
любой другой объект Java	значение, доступное только с помощью специального синтаксиса

Рисунок 7.1 – Правила типов

7.5. Пример

Код демонстрирует доступ к полям и методам Java из JSNI. Показывается, каким образом передаются числа, строки, логические переменные и объекты Java в JavaScript, а также вызов метода Java для переданного объекта:

```
public class J2JS {
	/** Pass a Java numeric primitive */
	public static void testJ2JSNumeric() {
		int x = 42;
		jsNumeric(x);
	}

	private static native void jsNumeric(int x) /*-{
		$wnd.alert("x is " + x);
	}-*/;

		/** Pass a Java String */
	public static void testJ2JSString() {
		String s = "my string";
		jsString(s);
	}

	private static native void jsString(String s) /*-{
		$wnd.alert("s is " + s);
	}-*/;

		/** Pass a boolean */
	public static void testJ2JSBoolean() {
		boolean b = true;
		jsBoolean(b);
	}
```

```
        private static native void jsBoolean(boolean b) /*-{
              $wnd.alert("b is " + b);
        }-*/;

              /** Pass an arbitrary Java Object */
        public static void testJ2JSObject() {
              MyJavaObject obj = new MyJavaObject();
              jsObject(obj);
        }

        private static native void jsObject(MyJavaObject obj) /*-{
              $wnd.alert("Calling getText(): " +
                    obj.@MyJavaObject::getTextAt(I)(3));
        }-*/;
}
```

Если посмотреть на исходные коды GWT можно обнаружить, что многое в нем определено как JSNI. Большинство программистов GWT никогда сами не пишут методы JSNI, но следует знать об их наличии. Следующая глава описывает языки и поддержки библиотек, реализованных в GWT на стороне клиента.

Глава 8
Эмуляция Java

Несмотря на то, что все приложение GWT может быть написано на Java, его части преобразуются в JavaScript для выполнения в браузере. С этим связана пара больших ограничений:

1. Клиентский код должен быть ограниченным подмножеством языка Java, которое поддерживается Google транслятором Java в JavaScript.
2. К тому же, код, выполняемый на клиентской части, может использовать только подмножество библиотек JRE, которое определено для JavaScript.

Как только вы привыкнете к этому, эти ограничения не будут настолько плохими, как кажется на первый взгляд. Это нечто подобное написанию мобильных приложений Java, где приходится ограничиваться определенным профилем, таким как MIDP. С помощью JSNI (глава 7) также можно расширить библиотеку с использованием native кода JavaScript, чтобы использовать любые области, которые вы посчитали недостающими.

8.1. Подмножество языка

Java в JavaScript GWT транслятор разбирает исходный код подобно тому, как это делает компилятор Java, но вместо компиляции в байт-код, транслятор генерирует код JavaScript.

Несмотря на содержании Java в имени, язык JavaScript достаточно отличается от Java. Например, он не использует классы и типизацию. Тем не менее, он достаточно гибкий, чтобы эмулировать множество библиотек Java 1.4.

Если вы используете возможности языка, которые не поддерживаются GWT, код должен нормально работать в hosted режиме, но при запуске компилятора Java в JavaScript для подготовки к web режиму, компилятор генерирует ошибку.

Уровень языка

GWT компилирует исходный код Java, который совместим с J2SE 1.4.2 или более ранней версией. Возможности языка Java 5, такие как улучшенные циклы `for`, шаблоны и аннотации, не поддерживаются текущей версией, но, возможно, будут поддерживаться в будущем.

Типы данных

Поддерживаются `byte`, `char`, `short`, `int`, `long`, `float`, `double`, `Object`, `String` и массивы. Тем не менее, в JavaScript нет типов

размерностью 64 бит, поэтому переменные `long` преобразуются в числа JavaScript с плавающей точкой удвоенной точности. Чтобы получить максимальную совместимость hosted и web режима, Google рекомендует использовать переменные типа `int`.

Исключения

Механизм `try`, `catch`, `finally` и пользовательские исключения поддерживаются обычным образом, но `Throwable.getStackTrace()` не поддерживается в web режиме.

Разрешения

Компилятор GWT разбирает конструкцию `assert`, но не генерирует для нее кода JavaScript. Разрешения обрабатываются в hosted режиме, если эта опция включена аргументом виртуальной машины.

Многопоточность и синхронизация

Интерпретатор JavaScript является однопоточным, поэтому, когда GWT применяет ключевое слово `synchronized`, от этого нет пользы. Библиотека для синхронизации методов недоступна, включая `Object.wait()`, `Object.notify()` и `Object.notifyAll()`.

Отражение

Для максимальной эффективности, GWT компилирует исходный код Java в монолитный скрипт и не поддерживает динамического создания классов. Эта и другие оптимизации исключают общее использование отражения. Возможно запросить объект по названию его класса используя `GWT.getTypeName()`.

Финализация

JavaScript не поддерживает финализацию объектов во время сборки мусора, поэтому в web режиме GWT не может использовать эту возможность. Большинство экспертов Java не советуют использовать финализацию, так что это не является большой потерей.

Ограничения плавающей точки

Спецификация языка Java точно определяет поддержку чисел с плавающей точкой и чисел с двойной точностью, а также ключевого слова `strictfp` и не может отличаться в преобразованном коде, поэтому следует избегать вычислений на клиентской стороне, которые требуют точности.

8.2. Подмножество библиотек

GWT поддерживает относительно небольшое подмножество библиотек JRE для клиентского кода. Одна из причин этого заключается в громадности библиотеки JRE, а другая в том, что многие из них не поддерживаются концепцией JavaScript. Так что забудьте о файловом вводе/выводе: его, например, можно сделать на сервере.

Далее представлены некоторые специфические методы, в которых эмуляция GWT отличается от стандартного выполнения Java. Для более подробной информации по определенным классам и методам, смотрите раздел 8.3, Поддерживаемые пакеты.

Регулярные выражения

Синтаксис регулярных выражений подобен, но не идентичен регулярным выражениям JavaScript. Вы, скорее всего, заходите использовать только регулярные выражения Java, которые имеют то же значение и в JavaScript. Подмножества регулярных выражений, которое поддерживают оба языка одинаково, описаны в разделе 8.4, Регулярные выражения.

Сериализация

Сериализация Java базируется на нескольких механизмах, которые не доступны в JavaScript, таких, как динамическая загрузка классов и отражение. В результате GWT не поддерживает стандартную сериализацию Java. Вместо этого у GWT есть возможность RPC (глава 5), которая обеспечивает автоматическую сериализацию объектов на сервер и обратно с целью выполнения удаленных методов.

8.3. Поддерживаемые пакеты

Для клиентского кода GWT реализует подмножество пакетов `java.lang` и `java.util` из JRE 1.4. Этот раздел перечисляет

поддерживаемые классы и интерфейсы, а также некоторые тонкости их использования.

Пакет `java.lang`

Поддерживаемые классы и интерфейсы для пакета `java.lang`:

Классы:
`Boolean`, `Byte`, `Character`, `Class`, `Double(1)`, `Float(1)`, `Integer`, `Long(1)`, `Math`, `Number`, `Object`, `Short`, `String(2)`, `StringBuffer` и `System`.

Замечания:
1. Избегайте использования как ключ словаря (производительность).
2. Регулярные выражения отличаются от стандартной реализации (раздел 8.4, регулярные выражения).

Ошибки и исключения:
`ArrayStoreException, AssertionError, ClassCastException, Error, Exception, IllegalArgumentException, IllegalStateException, IndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, NumberFormatException, RuntimeException, StringIndexOutOfBoundsException, Throwable(1)` и `UnsupportedOperationException`.

Замечания:
1. В текущей реализации не поддерживается трассировка стека.

Интерфейсы:
`CharSequence`, `Cloneable` и `Comparable`

Пакет `java.util`

Поддерживаемые классы и интерфейсы для пакета `java.util`:

Классы:
`AbstractCollection, AbstractList, AbstractMap, AbstractSet, ArrayList, Arrays, Collections, Date, HashMap, HashSet, Stack` и `Vector(1)`.

Замечания:
1. Не происходит проверка на правильность индекса.

Ошибки и исключения:
`EmptyStackException, NoSuchElementException` и `TooManyListenersException`.

Интерфейсы:
`Collection, Comparator, EventListener, Iterator, List, Map, RandomAccess` и `Set`.

8.4. Регулярные выражения

Как указывалось раннее, любой код на стороне клиента, использующий регулярные выражения, требует ограничений на выражения, которые понимают и Java, и JavaScript. Ниже приведены некоторые конструкции:

Символ

Выражение	**Значение**
\(обратный слеш)	следующий символ должен быть кодом
\cx	контрольный символ
\f	форматирование (‘\u0000C’)
\n	новая строка (‘\u0000A’)
\r	возврат каретки (‘\u000D’)
\t	табуляция (‘\u0009’)
\xhh	значение hh в шестнадцатеричном формате
\uhhhh	значение hhhh в шестнадцатеричном формате

Классы символов

Выражение	**Значение**
[xyz]	последовательность символов
[^xyz]	последовательность символов
\d	число
\D	не число
\s	пробельный символ
\S	непробельный символ
\w	цифра или буква
\W	не цифра или буква

Сопоставление границ

Выражение	**Значение**
^(символ)	начало строки (насколько я знаю, не поддерживается мультирежим)
$	конец строки
\b	граница слова
\B	не граница слова

Квалификаторы

Выражение	**Значение**
*	ноль или более, правильные все вхождения
+	один или более
?	ноль или один
{n}	точно n раз
{n,}	n или более раз
{n,m}	между n и m раз включительно

Разное

Выражение	**Значение**
.	любая переменная
(x)	захват группы
(?:x)	незахваченная группа
x|y	x или y
\n	обратная ссылка на захваченную группу (не используйте ‘\0’, потому что его значение в Java и JavaScript отличается)

Учитывая несколько простых ограничений, указанных в этой главе, можно рассчитывать на то, что Java код запустится в браузере. Код может распространяться на клиентскую и серверную часть программы, так что не следует реализовывать одинаковые алгоритмы на двух различных языках.

Итак, это все, что охватывает эта книга в области Google Web Toolkit. Надеюсь, эта информация была для вас полезной. Теперь заканчивайте читать и начинайте создавать что-нибудь великое!