Google Web Toolkit

Your Shortcut to

Ajax Web Applica

by Dan

GETTY IMAGES

"Web 2.0 is the future! I need you to build us a new Web front end for our site. It should use all the slick animations, auto-suggest, and desktop-like features that are the standard these days. It will need to be fast, and it has to work with Internet Explorer, Firefox, and Safari so we can support all our customers. How soon can you be done?"

Rachel broke out into a sweat remembering her boss's request. She had a lot of experience programming in Java and was a whiz at refactoring using Java IDEs—but Ajax? She was pretty sure that required knowing JavaScript and understanding the subtle quirks of all the popular browsers. How would she get through this?

Then Rachel remembered hearing about the Google Web Toolkit, which promised that you could build Ajax applications using Java instead of JavaScript and that it would handle all the complicated cross-browser issues automatically. She downloaded the Google Web Toolkit distribution, fired up her Java IDE, and started writing ...

Building Ajax applications can be tricky—just as juggling chainsaws while taming angry lions can be tricky.

Ajax application code is written in JavaScript, which then runs in a Web browser. A public Web site must support multiple versions of multiple browsers (Internet Explorer 5, 6, and 7; Firefox; Opera; and Safari), each with its own differences and idiosyncrasies. A developer has the difficult task of writing JavaScript that safely manipulates the Document Object Model (DOM) in a way that behaves identically on all supported browsers. Of course, code on the server side usually uses a different language and set of libraries, so there's a lot of context switching to be done. And we haven't even considered the hardest part—the application's real business logic!

Ajax is short for *asynchronous JavaScript with XML*. Ajax applications are highly responsive and feel more like desktop clients than traditional, form-based Web pages. The fundamental difference between standard Web sites and Ajax applications is that standard Web sites deliver all their data in a single page load, whereas Ajax applications may make several requests for data while the application is running and a single page is displayed. Data is delivered *asynchronously*, which means the user may continue to use an application while waiting for data from the server. This means the server components need only send the raw data and let the Web browser handle all the formatting and presentation logic.

Google has created an innovative tool to aid Ajax developers: the Google Web Toolkit (GWT). GWT compiles Java source code into browser-friendly JavaScript. This means you can use your favorite Java IDEs, design patterns, and tools such as JUnit, FindBugs, and PMD to create Web 2.0 applications. GWT includes a rich component library of user interface widgets, remote server communication protocols, and an emulated Java Runtime library, which maps Java types to JavaScript types—all of which work seamlessly across the major browsers. Furthermore, you can debug GWT applications using a standard Java IDE. These features let you focus on writing your application instead of worrying about browser compatibility problems. Using GWT is like having a teammate who knows about all the different

```
// This is a Java method
public int add(int x, int y) {
  return x + y;
}

// This is a JavaScript method
public native void alert(String msg) /*-{
  $wnd.alert(msg);
}-*/;
```

Listing 1

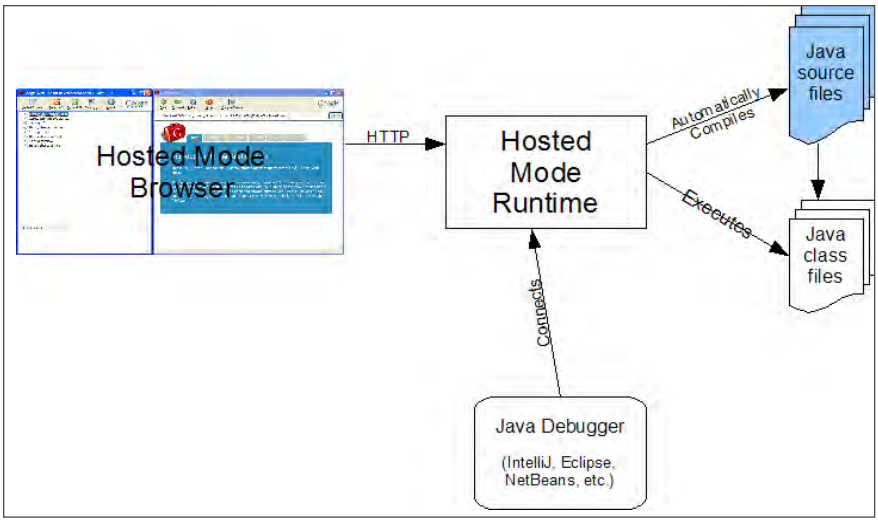

Figure 1: Hosted mode runs applications as Java bytecode and enables debugging using Java IDEs.

browser incompatibilities and handles them for you.

Java to JavaScript Compiler

What makes GWT unique is its Java-to-JavaScript compiler. You write plain old Java code and invoke GWT's compiler, which takes Java source files and outputs JavaScript. This is a true compiler—not a simple text transformation step like Ruby on Rails' RJS, which provides Ruby objects that use String manipulation to create JavaScript output. While Java is the preferred language for writing GWT applications, it doesn't mean that you have to give up using JavaScript. You may mix Java and JavaScript code in the same class with GWT's JavaScript Native Interface (JSNI) [1] by using the `native` keyword.

Listing 1 demonstrates a Java class with a native JavaScript instance method. You can pass objects back and forth across the Java/JavaScript boundary and generally treat the code as if it were all part of the same object. Using Java as much as possible to ensure browser portability is a best practice, as it leverages all the GWT safety features, such as preventing memory leaks. However, if you need it, you can always write JSNI methods.

Optimizations

Another benefit of using a compiler is that it allows for compiler optimizations both in terms of performance and generated JavaScript source file size. The GWT compiler can replace many levels of delegation with inline method calls, which can yield code that's as fast as code that has been hand-optimized. This means you can focus on writing well-factored, object-oriented code instead of obsessing about performance tweaks.

Ajax applications should be fast and responsive, which means minimizing initial page load time. One way to do this is to simply send less JavaScript code from the server to the client. The GWT compiler can create compact, obfuscated JavaScript optimized for short download time or human-readable code, which is useful when diagnosing problems. In addition, GWT analyzes your entire application and only includes the code that is actually used. Practically, this means that you can reuse existing Java libraries and other modules, and GWT will compile only the routines you actually invoke. This optimization comes at a price; you can't use reflection in your code since GWT can't tell ahead of time what methods are really invoked.

Debugging and Testing

Some problems are easiest to diagnose with a debugger, and GWT makes debugging easy, as shown in figure 1. Simply start up your application in "hosted mode," which runs your application using Java bytecode instead of JavaScript. This means you can use your Java IDE's debugger just as you would for server-side code. To run applications in hosted mode, the GWT development kit includes an embedded browser instance of Internet Explorer for Windows, Firefox for Linux, and Safari for Mac. Hosted mode is really useful, and you'll probably end up using it for most of your development work with GWT. For example, you can run your application, change the source code, and hit the browser's refresh button to see your changes immediately. Hosted mode also provides a console window that you can use to log debug messages during development.

You can write unit tests with JUnit just as you would for server-side code. However, when you need to test applications that manipulate the DOM or use JavaScript, GWT can run your unit tests inside the same hosted-mode browser you use for developing applications. If you subclass GWTTestCase instead of the JUnit standard TestCase, your test will run in an invisible browser with access to the DOM. You can run these tests as compiled Java bytecode or as JavaScript to ensure your application behaves correctly after being compiled.

Component Libraries

GWT includes a rich set of class libraries and user interface widgets. A subset of the core Java SE libraries is emulated by GWT, which means you can use most of the core language data types and collections. GWT also provides a rich set of browser widgets as well as utilities for supporting multiple languages and a way to handle the browser's back and forward buttons.

Widgets and UI Components

As figure 2 shows, Google provides many widgets with GWT—from simple, standard HTML widgets like buttons and text fields to rich widgets composed of simpler elements like a tree view. Simple widgets such as `Button`, `Checkbox`, `Hyperlink`, and `TextField` provide simple wrappers around basic HTML-form elements and are used for the majority of form-based data input. For more dynamic interfaces, GWT offers a rich set of widgets including: `TabPanel` for navigating through a tabbed set of pages; `SuggestBox` for a text entry field, which lists possible completions

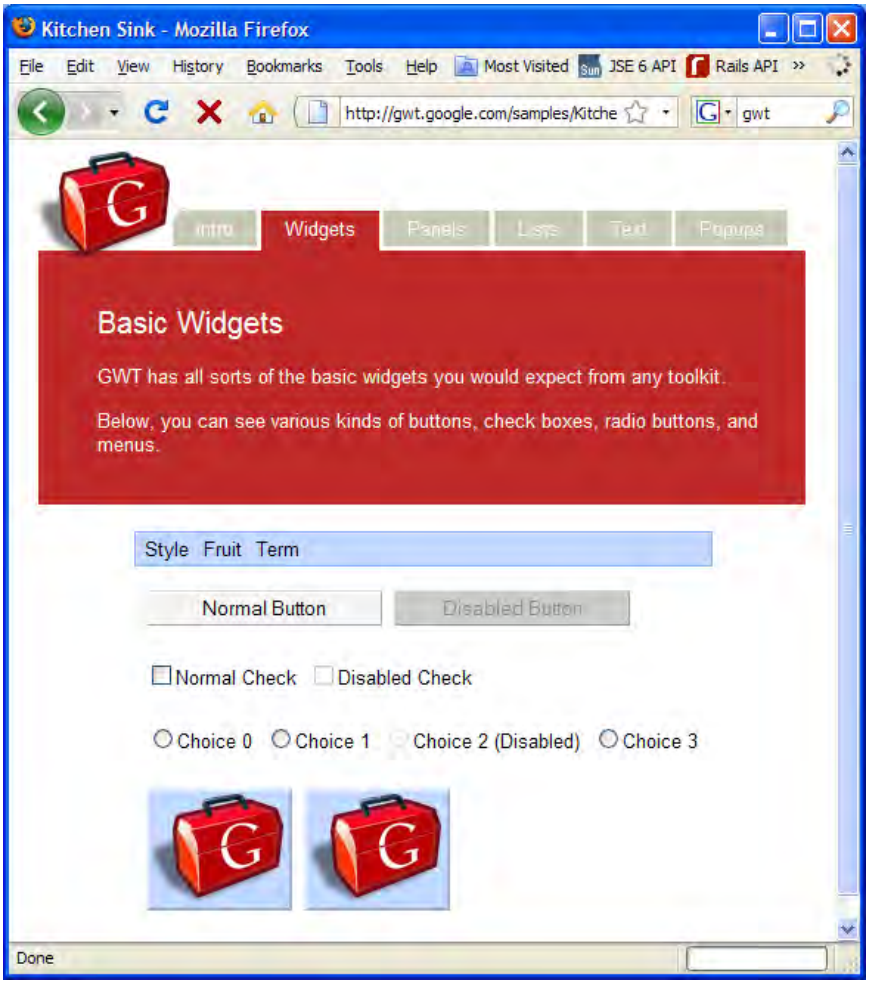

Figure 2: GWT provides many widgets from simple buttons to tabbed menus and dialog boxes.

as you type (think Google Suggest); and `DialogBox` for dragable popup windows.

GWT applications use an event-based programming model, where each button, text field, and other user interface element responds to a set of actions such as clicks, content changes, and focus changes. This programming model will be familiar to developers who have written Swing desktop applications. In fact, GWT code looks similar to Swing code. You add behavior to your widgets by adding event listeners that respond to user interactions. For example, listing 2 shows how you define a button that prints a message every time you click it.

By building your applications using GWT widgets and the event-based programming model, you let GWT handle the complex differences between browsers. For example, did you know that Internet Explorer versions 5 and 6 have a problem where select boxes appear through any covering popup windows? GWT prevents this display glitch by applying a well-used fix that creates IFRAMEs that sit directly between the select box and the popup window. Did you know that Internet Explorer propagates events through the document model differently from all other browsers? By using GWT's event classes, you never need to know about or deal with the differences.

Internationalization Support

GWT's built-in internationalization features can help you build a multi-language site. Your application's text and messages can be extracted into a separate property file for each supported language. GWT will then compile these property files and generate separate versions of your Web application for each language. Dates and currencies can be formatted for display in a given locale by using the `DateTimeFormat` and `NumberFormat` classes.

Back Button and Browser History

One of the most difficult issues in writing Ajax applications is handling the browser's back button. An Ajax application is usually a single page that requests new data from the server without navigating to another page. Since browsers track navigation history by full-page loads, handling the back button is tricky. For example, consider working with the Ajax-heavy Google Maps site: A user navigates to Google Maps from the Google start page then searches for the closest pizzeria. The user finds several results and clicks through a few choices to view their details. When the user hits the back button, he expects to go to the previously viewed pizzeria—not back to the Google start page.

GWT also provides a history mechanism for creating applications that respond predictably to the back and forward buttons. This mechanism relies on a common JavaScript trick that uses an IFRAME HTML element to generate page history. All you need to do for your application is add an IFRAME element to your page and let GWT handle all the JavaScript tricks for you. In your application, you tell GWT when to add an entry into the browser history, such as clicking a link or button. When the user presses the back button, your application will be notified and you can change your page as needed.

Remote Server Communication

Ajax applications wouldn't be very interesting if they didn't have access to data. GWT provides a number of ways to access data on remote servers, including a custom protocol that lets you easily use your server-side objects directly in the browser. To access data services from other applications, GWT also supports standard data interchange formats like JSON and XML.

Using Server Objects on the Client with GWT RPC

If you're building both the server side and the client side of a GWT application,

```
Button button = new Button("Click Me");

button.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
            Window.alert("I was clicked!");
      }
});
```

Listing 2

then you'll get a big productivity boost by using GWT's Remote Procedure Call (RPC) feature. This mechanism allows you to use your server-side objects directly on the client side. Consider what would be required to use a plain-text format: You'd fetch the required data on the server side, write code to turn your objects into a text representation, send this text to the client, then write code on the client that can parse and interpret the message. With GWT RPC you can skip writing extra encoding and decoding layers and start actually *using* the data.

To use the GWT RPC feature, you can use GWT's servlet-based RPC framework for new applications or use the RPC libraries directly with your existing Web application frameworks, such as Spring and Struts.

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Widget",
        "name": "main_window",
        "width": 500,
        "height": 250
    },
    "flags": [ 1, 8, 16 ]
}}
```

Listing 3

Communicating with Other Back Ends with HTTP, JSON, and XML

Just because GWT client-side code is written in Java doesn't mean that the server must be Java as well. The `RequestBuilder` class can be used to communicate with any back end: Python, C#, or any language that can speak HTTP.

If you aren't using GWT RPC in Java or have a server back end in another language, GWT supports the standard XML and the JavaScript Object Notation (JSON) data interchange formats. JSON has been gaining popularity in Web services and is easily read by humans and browsers; in fact, it's a valid JavaScript object, as shown in listing 3.

JavaScript and GWT Third-Party Libraries

Successful development platforms have a rich community of third-party libraries, and GWT can use both JavaScript and other GWT libraries.

Since GWT applications are always deployed as JavaScript, you can easily use other JavaScript libraries. This means you can use any existing JavaScript code your company may have developed, as well as popular third-party tools, such as the animation framework Scriptaculous and other application frameworks like ExtJS. You'll need to write native JavaScript code, which acts as a bridge between your Java and JavaScript code. However, many folks have already done this work for you and published these bridges as GWT libraries.

In addition to JavaScript library wrappers, there are many useful open source components made specifically for GWT. GWT-Maven provides support for the popular Maven build tool to build and package GWT applications. Gwittir offers an alternative to reflection to provide simple JavaBean-style introspection that can be used for data binding and validation. Popular server-side frameworks are made GWT-friendly through projects like Hibernate4gwt, which allows you to use Hibernate objects easily in your client code, and GWT-SL, a server-side library that provides Spring framework support. Check out the GWT Google Group for discussions and announcements of other tools.

Google also publishes additional libraries for GWT. It wouldn't be a Google project if you couldn't access the Google online services, and the Google API Libraries for GWT project does just that: You get access to Maps, Search, offline functionality with Gears, and the multi-platform Google Gadgets library for desktop and Web widgets. In the eating-their-own-dog-food category, the gwt-api-interop library was used with GWT to help build the iPhone Google Reader application. It provides a way to bind JavaScript objects to Java objects, which means you can bind JSON to Java objects without writing a parser by hand.

The Real World

The story at the beginning of this article isn't pure fiction; a similar situation happened to me on a recent project. My team was hired to build the front end to a new site, and we had used GWT previously. We had to support the major browsers on Windows and Mac and deal with requirements that were evolving throughout the four-month project. GWT allowed us to use our Java experience with test-driven development and refactoring to keep our design clean and deliver on time. We started with two full-time developers and scaled up to four, then six—including two new hires. GWT's simple programming model made this an ideal first project, as the team spent more time focusing on the business rules than learning APIs.

The next article in this series will focus on testing and GWT, including how to build Ajax applications test-first to help create simpler, defect-free designs. **{end}**

REFERENCES:

1] "JavaScript Native Interface (JSNI)" in the Google Web Toolkit online documentation. code.google.com/webtoolkit/documentation/.

Sticky Notes

For more on the following topics go to www.StickyMinds.com/bettersoftware.

- More information
- Other Ajax framework options