CHAPTER 1

Web 2.0 and Struts2

Before charging forward with developing a Web 2.0 application, you need to understand what a Web 2.0 application really is. In this chapter, you will learn what Web 2.0 means from a development as well as end user perspective.

With Struts2 being the technology of choice, you will also learn how Struts2 provides the features to make developing a Web 2.0 application easy.

What Is Web 2.0?

One of the questions that needs to be answered before embarking on developing a Web 2.0 application is "What is Web 2.0?" As it turns out, this is a particularly difficult question to answer.

From a programming perspective, Web 2.0 is synonymous with AJAX (Asynchronous JavaScript and XML). The term AJAX was coined in February 2005 by Jesse James Garrett and is used to describe the interaction between many technologies. At the core is the `XMLHttpRequest` object, which is supplied by the web browser. This object was first present in Microsoft Internet Explorer 5 (released in March 1999), although similar techniques using `IFRAMES` and `LAYER` elements have been available since 1996.

Along with the `XMLHttpRequest` object, the technologies that make up an AJAX interaction are the following:

- *HTML/XHTML (Hypertext Markup Language)*: Used to present information to the user from within the web browser.
- *DOM (Document Object Model)*: The object structure of the HTML document in the web browser. By manipulating the DOM with JavaScript, the page rendered to the user can be modified dynamically without reloading the current page.
- *CSS (Cascading Style Sheets)*: Used to format and style the HTML presented. By separating formatting from structure, the code can be modified consistently and maintained more easily. Similarly to the DOM, CSS for the current page can be modified via JavaScript to dynamically change the formatting without reloading the current page.

- *JavaScript*: A programming language that can be embedded within HTML documents. JavaScript code or functions can be executed inline (as the page is processed), in response to HTML events (by providing JavaScript in the value of HTML attributes), or triggered by browser events (for example, timers or user events).
- *XML (eXtensible Markup Language)*: The format of the data returned by the server in response to the asynchronous call from the web browser. The XML response returned is processed by JavaScript in the web browser, causing changes in the HTML (by manipulating the DOM or CSS).

Recently, another data format has been gaining popularity: JSON (JavaScript Object Notation). Similar to XML, JSON returns data that can be processed within the web browser using JavaScript. The advantage of JSON is that it can be very easily parsed by JavaScript; in fact, to convert a response of any size from the JSON transport format to JavaScript objects involves a single call of `eval('('+responseJSON+')')` (where `responseJSON` is the JSON data represented as text or a string). Using JavaScript to process XML is much more involved and requires at least one line of code to assign a value from the XML document to a JavaScript object.

EVALUATING VS. PARSING

There is a security concern when calling `eval()` on a JSON string, especially when the JSON is obtained from a source external to the code currently being executed. The problem lies in the fact that the `eval()` function compiles and executes any JavaScript code in the text string being parsed to create the object representation. For this reason, you need to be sure that you trust the source of the JSON text. Even better still, you can use a JSON parser, which avoids the problems associated with the `eval()` function.

One such parser can be found at `http://www.json.org/json.js` (the web site `http://www.json.org` is the gateway to all things JSON). When using this JavaScript script, additional methods are added to the basic JavaScript objects to both generate JSON and parse JSON. When provided with a JSON string to be parsed (say `jsonText`), the following code is used:

```
jsonText.parseJSON(filter);
```

The parameter `filter` is an optional JavaScript function, which can be used to further filter and transform the result. To generate JSON, use the `toJSONString()` method. For example, to convert a boolean `myBoolean`, use the following:

```
myBoolean.toJSONString();
```

By using a JavaScript JSON parser, the JSON text can be converted just as simply but without security concerns.

By using AJAX interactions, developers can make the user experience less awkward. Rather than requiring the entire HTML page to be reloaded from the server (along with processing the request on the server) and rerendered to update values in a drop-down selection box, now a smaller request to the server can be made. More importantly, the page is not rerendered; instead, the only change to the HTML is that the values for the drop-down selection box have now been changed.

Smaller and more targeted information requests to the server means that the time spent waiting for the network and server processing will be less. Not having to rerender the entire browser page on each server request will also be perceived as the web application performing faster. With these pieces working together in an AJAX interaction, the web browser will become more responsive and act more like a traditional desktop application—increasing the usability and overall user experience.

It also means that developers need to think differently. In fact, developers need to reexamine the fundamental way that a web application is constructed; rather than thinking of a page as a unit of work, they need to think of functionality from a page as being the unit of work, with many functions being combined to create the final page. Furthermore, the same functionality can now be easily shared among pages.

THE PAVLOV EFFECT

Changing the user interaction (even for the better) has its own problems. Users have been trained to understand that nothing on HTML pages changes until you click a link or a form submit button. All of a sudden, things are different. Now, at any time, any part of the HTML page has the potential of being updated or removed, and new information can be added.

To help transition users to the new browser interaction model, as well as to provide developers with guidelines of when and how to use AJAX in web applications, a series of patterns has emerged. AJAX patterns cover a wide range of topics, including how to signal the user that a UI element has changed; when to make server calls to obtain data; options for introducing AJAX into non-AJAX web applications; how to manage technical aspects such as server timeouts; and ways to provide a non-AJAX fall-back when JavaScript is not available on the user's browser.

From a marketing or end-user perspective, things are a little different. There is no doubt that more interactive user interfaces can make the overall web application's usability better, however, the shift from Web 1.0 to Web 2.0 is more than user interfaces.

In September 2005, Tim O'Reilly published an article titled "What Is Web 2.0" (`http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`). This article explored what Web 2.0 was by comparing a new breed of web sites that were available to those that had been around for some time. The result was that no hard boundaries of principles or technologies signified an application as a Web 2.0 application. Instead, there were guiding principles that, when adopted, resulted in a web application that is more Web 2.0 than when the principles were not used. Following is the list of proposed principles:

- *Web as a platform*: Applications should take advantage of the Web as a platform rather than simply providing a presence on the Web. By working symbiotically with the openness and connectedness of the Web, services can reach out to all users. And in doing so, will get better as more people use the service.

- *Harness collective intelligence*: Hyperlinking has been the foundation of the Web, allowing users to explore related content, but it has always been provided by the web site being visited. The new breed of applications takes this a step further, allowing users to provide information to the web application in the form of content and metadata (information about the content, such as ranking or popularity). Individual publishing, via blogging, has also become popular, allowing anyone to become a publisher of information and opinions.

- *Data is the next "Intel inside"*: Originally, companies owned and provided data to users of their application. Although an initial set of data is still this way, a new and much more valuable set of data is being provided by users of the application. Now the race is on for companies to own a particular category of user-provided data, leading to the question, "who owns the data?"

- *End of the software release cycle*: With software being delivered as a service rather than a product, all users of a web application can take advantage of new features being provided immediately. In a sense, users then become codevelopers and can be monitored to determine which features are used, and how often—shaping the features of the final product. Releasing often also requires operations to become a core competency.

- *Lightweight programming models*: There is a preference to use simple protocols, such as REST (Representation State Transfer) rather than SOAP (Simple Object Access Protocol), and RSS (Really Simple Syndication) to provide syndication and remixability. The innovation is that combining many loosely coupled services together in a unique or novel manner provides value in the assembly.

- *Software above the level of a single device*: In a connected world, the browser is no longer the single device of choice. Web applications will need to interact with devices accessing them from web browsers and cell phones, as well as more specialized devices such as iPods, PDAs, and cable boxes.

- *Rich user experiences*: Services are becoming RIAs (Rich Internet Applications), providing dynamic and responsive interactions with users. AJAX (which was explained earlier) as well as Java applets and proprietary solutions such as Macromedia Flash, are all enabling technologies.

Almost one year later in August 2006, Tim O'Reilly gathered a group of people together to build on his initial paper. Gregor Hohpe was one of those people invited, and he blogged (`http://www.eaipatterns.com/ramblings/45_web20.html`) about the values, principles, and patterns that were discussed.

As an agile developer, the style the values were presented in hit an accord. Using the same format as the Agile Manifesto, it presented the differences between a Web 1.0 and Web 2.0 application as a range. The closer the application is represented by the descriptions on the left, the more Web 2.0 the web application is. In the end, whether an application is Web 1.0 or

Web 2.0 is still subjective, but grading the level of Web 2.0-ness is easier. The values, with my interpretation, are provided here:

- *Simplicity over Completeness*: Application features do not need to be absolutely complete, having every variation and every option possible. Instead, the most used options are all that is required, making the application much simpler to use and quicker to market.
- *Long Tail over Mass Audience*: Business models are focusing on selling smaller volumes of a large variety of hard-to-find or unique items rather than selling large volumes of a small number of popular items. The same can be said about knowledge (see the Wikipedia entry for more information on the Long Tail `http://en.wikipedia.org/wiki/The_Long_Tail`).
- *Share over Protect*: Web sites are no longer gated enclosures; instead, information and services are shared using techniques such as web services and feeds.
- *Advertise over Subscribe*: The preferred revenue model for Web 2.0 sites is advertisement rather than subscription (although of all the values, this is the one that is most controversial because as applications move from products to Web 2.0 services, a subscription model will be required).
- *Syndication over Stickiness*: An early goal of web applications was to keep users on the site for as long as possible. By providing services, the information that could only reach users on the site can now have a much farther reach by syndication (with links leading them back to the application).
- *Early Availability over Correctness*: Rather than working behind closed doors to perfect a web application feature, it's more important to get the features out to users so they can assist as codevelopers in the perfecting the features.
- *Select by Crowd over Editor*: The opinions and aggregated wisdom of many people is far more valuable than the opinion of a single person.
- *Honest Voice over Corporate Speak*: The opinions of experts participating in or using a service or product are more valuable than marketing information that has no personal insight.
- *Participation over Publishing*: Whenever possible, it's better to allow the users to participate and share their experience, rather than publishing edited information.
- *Community over Product*: Creating a community and then taking advantage of the collective knowledge of the community is more important than providing a product with individual user access.

The interesting thing is that in this second phase of the Web, the focus is once again on collaboration and sharing information and opinions. This was an original goal of the Internet (`http://en.wikipedia.org/wiki/History_of_the_Internet`) when universities were exploring ways to collaborate.

Web Application Development 2.0

After reviewing the values and principles that make up a Web 2.0 application, you might be asking yourself "how is this different from what I am doing now?" We have reviewed AJAX interactions in the previous section, and this is by far the most significant change from a development perspective. Other changes are at a far more fundamental software development level and less visible to the end user:

- *Development process agility*: As a service, software features can be changed at lightning speed. It could be at a client's request or as new business requirements are introduced, but either way, a process must be in place to efficiently introduce new features and validate that the new code has not broken existing features. More than ever, unit testing, continuous integration, and automated deployment processes are required to support the development efforts.
- *Syndication and integration*: Two sides of the same coin, syndication and integration allow your application to share data with other external applications as well as use services from external sources. When architecting your web application, thought needs to be put into determining how the application will technically achieve these objectives, as well as what format the data and services being provided will take.
- *Web framework agility*: Having a web development environment that works with the developer to provide an environment that is flexible, productive, and encompasses the values of Web 2.0 is of utmost importance. With Web 2.0, there has been a resurgence of development in existing dynamic languages, such as PHP, as well as newer languages and frameworks, such as Ruby and Ruby on Rails. Struts2 is one of many Java frameworks that provide the maturity, experience, and features to compete with dynamic language frameworks.

The features listed previously are not technical features of web development frameworks, and this is important. As web development matures into a second phase of growth, the focus is on business models and features provided to the users. Technically, the difference is on how the applications are developed—by integrating services (that may be provided by other applications, known as mashups) and data together to provide value.

Web Framework Agility with Struts2

Because the focus of this book is on web development, we will explore how Struts2 provides agility as a web application framework. However, before getting to Struts2, we need to talk briefly about a new web framework that made its debut around the same time that web applications were releasing Web 2.0 features. This framework is Ruby on Rails.

When Ruby on Rails was released in August 2004, many (if not all) existing web application frameworks went through a period of self-examination; new frameworks were also created (Grails, for example). Several driving factors made Rails so compelling to use as a developer:

- *Full web application stack*: All the basic elements necessary to build a web application were provided in the base distribution.
- *Convention over configuration*: Rather than configuring every element of the application, conventions were used (that could be overridden). A standard directory structure (where each development artifact had a known location) and standard naming conventions are a large part of the conventions.
- *Scaffolding*: The framework can provide fully functional basic user interfaces (with controller logic) for model objects, allowing the application to be used while the real production code is being developed.
- *Interpreted development language*: The underlying development language is Ruby, which is dynamic, object-oriented, and interpreted.

All these features allow developers to be more productive from the initial download and setup of the framework, to the day-to-day development of new web application features.

Struts was first released in July 2001 and was an overwhelming success. It provided an MVC (Model View Controller) pattern implementation for Java web development, allowing web applications written in Java to segment code (rather than writing HTML code in servlets or Java code in JSPs) and to manage reusability and maintenance of existing code.

Over time, developing in Struts required more developer-provided code to implement the necessary web application features, and a proposal for the next generation of Struts was suggested. Architecturally, the changes necessary to implement the proposed features were significant and, rather than starting from scratch, Struts developers approached other open source Java frameworks with a proposal for a merger. Without covering all the details, the result was that WebWork (an OpenSymphony project that itself was an early fork of the Struts code base) merged with Apache to become the basis of Struts2.

■ **Note** The history of Struts Ti and the WebWork/Struts merger is documented by Don Brown at `http://www.oreillynet.com/onjava/blog/2006/10/my_history_of_struts_2.html`.

Interestingly, one of the WebWork lead developers, Pat Lightbody, had been reviewing features of Ruby on Rails with the goal of making WebWork more productive and easier for developers to use. Some of these features are now part of the Struts2 feature set, and some (because of the Java language constraints as well as maturity reasons) did not make the transition.

Following is a list of Struts2 features that drive the framework forward to be more developer friendly and productive:

- *Java*: The Java language has been around for 10 years now and has matured to a point where most of the features (nonbusiness-related) already exist as libraries. Java is typed (a plus for some developers) and can access an organizational infrastructure that has already been developed (although JRuby and Groovy have options for calling existing Java classes via dynamic languages).
- *Plug-ins*: Functionality provided as core or third-party plug-ins can be added as needed, rather than requiring the core framework to include everything. As well, the plug-in development life cycle (and hence the introduction of new features) is no longer tied to the core framework, allowing more frequent upgrades.
- *Convention over configuration*: Wherever possible, configuration has been eliminated. By using zero-configuration, class names can provide action mappings, and returned result values can provide names for JSPs to be rendered.
- *Annotations rather than XML configuration*: There are two benefits to using annotations: first is a reduction in XML configuration, and second is the configuration is closer to the action class (reducing the effort necessary to determine the action configuration).
- *Data conversion*: The conversion of string-based form field values to objects or primitive types is handled by the framework (and vice-versa), removing the necessity of providing this infrastructure code in the action class.
- *Dependency injection*: All the objects that the action needs to interact with to perform the logic of the action are provided via setters. This reduces the coupling between application layers, and makes the application simpler and easier to test.
- *Testability*: Testing Struts2 actions, interceptor, and other classes is very easy.
- *MVC pattern*: Struts2 continues to use the MVC pattern, providing a layer of abstraction between the view (usually rendered as HTML) and the framework by using a URL. This is important because it doesn't tie the framework to a particular device or rendering style (such as, always refresh an entire page; partial HTML updates; and processing events supplied as AJAX requests).

A lot of work is still needed to get to the productivity level that Ruby on Rails is today, and some features will just never make it (due to the restrictions of the Java language). However, in choosing a web application framework, many factors are involved, and the selection of a programming language that can take advantage of the organization infrastructure that is already in place is one of the most important. With numerous options available to choose from for Java web application frameworks, Struts2 is just as strong of a contender today as it was when Struts was first released—providing the developer productivity features and Web 2.0 functionality that is needed to develop today's web applications.

Using this Book

Throughout the course of this book, the Struts2 framework will be used to develop a Web 2.0 application. As we have already discussed, Web 2.0 characteristics mostly focus around business features and the underlying business model of the organization. However, to develop a

fully featured application, you need to understand the framework, concepts, configuration, and the (non-Web 2.0 specific) features. With this in mind, this book is divided into four sections:

- Chapter 2 and Chapter 3 provide the fundamentals on Struts2 with information on how to get up and running, how a request is processed, background information on the framework, and configuration information and extension points.
- Chapter 4 provides the background information on the application that is to be developed throughout the course of the book, including the development process to be used, an overview of the application, the use cases that will be developed, and supporting technologies (that are used in combination with Struts2).
- Chapters 5 through 8 describe the core features of any web application: data manipulation, wizards and workflows, security, and rendering information.
- Chapter 9 and Chapter 10 focus on the Web 2.0 features of the application, including syndication, integration, and ways that AJAX can be integrated into the application.

■ **Note** The code was developed using Struts version 2.0.9. In most cases, the provided code should be compatible with any 2.0.x release; however, if you do choose to use a more recent version, there may be some incompatibilities.

Because the concepts and features being introduced are built upon the knowledge from previous chapters, reading the book forward from start to finish is recommended. If you are familiar with Java web application frameworks and don't want to read the entire book, start with Chapter 2 and Chapter 3. These provide the necessary Struts2-specific information so that you can pick and choose the other chapters to read. If you are familiar with Struts2 and are looking for specific implementation information, Chapters 1 through 4 can be safely skipped.

Finally, the application developed in this book illustrates the most common technologies used when developing web applications today: JSPs, the Spring Framework, and JPA. By using Struts2, you have many other options for the view, business tier, and data tier. Plug-ins are the most common mechanism for integrating new technologies, and the list of all the current plug-ins can be found at `http://cwiki.apache.org/S2PLUGINS/home.html`. When considering alternatives, this is the first place you should look.

The other reference you should keep handy is the Struts2 official document site: `http://struts.apache.org/2.x/docs/guides.html`. Here you will find the most up-to-date information on Struts2, as well as reference documentation, guides, tutorials, and release notes for released versions.