Struts is the premier framework for building Java-based Web applications. Using the Model-View-Controller (MVC) design pattern, Struts solves many of the problems associated with developing high-performance, business-oriented Web applications that use Java servlets and JavaServer Pages. At the outset, it is important to understand that Struts is more than just a programming convenience. Struts has fundamentally reshaped the way that Web programmers think about and structure a Web application. It is a technology that no Web programmer can afford to ignore.

This chapter presents an overview of Struts, including the historical forces that drove its creation, the problems that it solves, and the importance of the Model-View-Controller architecture. Many of the topics introduced here are examined in detail by subsequent chapters.

A Brief History of Web Application Development

In order to fully understand and appreciate the need for and value of Struts, it’s necessary to shed some light on how Web application development has evolved over the past several years. Initially the Internet was used primarily by the academic and military communities for sharing research information, most of which was in the form of static documents. Thus, originally, the Internet was mostly a mechanism for sharing files.

In 1995 the commercialization of the Internet began and there was an explosion of content made available on the Web. Similar to the research content that was being shared on the Web, the early commercial content was principally comprised of text mixed with simple graphics. Hyperlinks were used to connect the content together. Although hyperlinks enabled the user to move from page to page, the contents of each page was still a static document that did not support other forms of user interaction. It wasn’t long, though, before businesses wanted to be able to offer dynamic content that offered the user a richer, more interactive experience.

Before continuing, it will be helpful to explain precisely what is meant by dynamic content. In short, dynamic content is data that is specifically targeted for a particular user. For example, a user may want to check the price and availability of some item in
an online store. The user enters the item name and the server supplies the response.
The response is generated on-the-fly based on the request, and is thus dynamic content.

To fill the dynamic-content void, Web server software began to support the use of CGI scripts for creating applications that could run on a Web server and generate dynamic content back to a browser. CGI, or Common Gateway Interface, allowed Web servers to accept a request and execute a server-side program that would perform some action and then generate output on standard out. Web server software would then read that output and send it back to the requesting browser. Initially, many of these CGI scripts were written in Perl or other Unix-based scripting languages. Over time, though, as the applications being built to run as CGI scripts grew in complexity, more application-oriented languages like C and C++ were being used to create larger, more robust applications. With the advent of HTML forms, CGI scripts also were able to receive data from the browser and process it. As most readers know, HTML forms allow data entry on a Web page. That data could be sent to a CGI script on the server and then manipulated, stored, or otherwise processed.

Around the same time that CGI-based application development was becoming popular on the server side, the Java programming language was introduced, with an initial focus on applets. Applets gave the Web developer the ability to add rich, dynamic functionality to Web pages. Because Java offered the promise of “write once and run anywhere” programs, any browser that supported Java could run the applets. For the first time, developers could easily include dynamic content on a Web page.

For the same reasons that Java began to blossom on the client side with applets, Java also began to make inroads on the server side with the advent of servlet technology in 1997. Servlets solved many of the shortcomings of CGI, such as portability and efficiency, and offered a Java-based solution for the Web application paradigm. Servlets are portable across operating systems and can run on any server that has a Java Virtual Machine (JVM). Thus, they also benefit from Java’s “write once, run anywhere” philosophy. Servlets have a more efficient execution model than CGIs because they are multithreaded instead of requiring a new process for each request. Servlets also have access to Java’s vast libraries, including the JDBC APIs.

After servlets were introduced, Sun released the JavaServer Pages (JSP) technology as an extension to the servlet technology. JSPs take the reverse approach from servlets to building Web applications by having Java code intermingled in an HTML-based page. When a request is made to the server for a JSP, the Java server container checks if the JSP has already been compiled into a servlet. If it has, it proceeds to execute the servlet. If the JSP has not yet been compiled into a servlet, the server container converts the JSP code into a Java source file and then compiles that source so that subsequent requests to the JSP will find the servlet already compiled and ready to execute.

The nice thing about this approach is that changes to the JSP HTML can be made without having to manually recompile the code. The server container manages the
compilation and will recognize that the HTML in the JSP has changed and recompile
the JSP into a servlet for you. JSPs solve the problem of presentation code (HTML)
being embedded in servlets, which made development cumbersome because HTML
authors had to wade through Java code to edit HTML (not a good separation of
responsibilities). In contrast, HTML developers can work on JSPs directly without
interfering with Java code.

As the preceding discussion shows, many of the changes in Web-based development
that have occurred over the past few years have been driven by the desire to efficiently
include dynamic content in a Web page. Streamlining the use of dynamic content has
been, and remains, one of the more important issues associated with the Internet and
the applications that use it. As you will see, Struts is part of the solution to the dynamic-
content problem.

**Two Development Models**

When Sun introduced JSP technology, it provided a development road map for working
with it and defined two models for building JSP-based Web applications. The two models
are known as **Model 1** and **Model 2** and they prescribe different approaches to designing
JSP-based Web applications. Model 1, the simpler of the two, was the primary solution
implemented when JSPs were first introduced. However, over time, Model 2 has been
accepted as the best way for building JSP-based Web applications and, as you’ll see, is
the inspiration for MVC-based Web frameworks like Struts. Following is an overview of
both architectures.

**Model 1 Architecture Overview**

The Model 1 architecture is very simple, as you can see in Figure 1-1. A request is made
to a JSP or servlet and then that JSP or servlet handles all responsibilities for the request,
including processing the request, validating data, handling the business logic, and
generating a response. Although conceptually simple, this architecture is not conducive
to large-scale application development because, inevitably, a great deal of functionality
is duplicated in each JSP. Also, the Model 1 architecture unnecessarily ties together
the business logic and presentation logic of the application. Combining business logic
with presentation logic makes it hard to introduce a new “view” or access point in an
application. For example, in addition to an HTML interface, you might want to include
a Wireless Markup Language (WML) interface for wireless access. In this case, using
Model 1 will unnecessarily require the duplication of the business logic with each
instance of the presentation code.
Model 2 Architecture Overview

Model 2, or as it is most commonly referred to today, Model-View-Controller (MVC), solves many of the inherent problems with the original Model 1 design by providing a clear separation of application responsibilities (see Figure 1-2). In the MVC architecture, a central servlet, known as the Controller, receives all requests for the application. The Controller then processes the request and works with the Model to prepare any data needed by the View (which is usually a JSP) and forwards the data to a JSP. The JSP then uses the data prepared by the Controller to generate a response to the browser. In this architecture, the business and presentation logic are separated from each other. Having the separation of business and presentation code accommodates multiple interfaces to the application, be they Web, wireless, or GUI (Swing). Additionally, this separation provides excellent reuse of code.
A Closer Look at the Model-View-Controller Architecture

Because an understanding of the Model-View-Controller architecture is crucial to understanding Struts, this section takes a closer look at each of its parts. As a point of interest, MVC is based on an older graphical user interface (GUI) design pattern that has been around for some time, with its origins in the Smalltalk world. Many of the same forces behind MVC for GUI development apply nicely to Web development.

Model Components

In the MVC architecture, model components provide an interface to the data and/or services used by an application. This way, controller components don’t unnecessarily embed code for manipulating an application’s data. Instead, they communicate with the model components that perform the data access and manipulation. Thus, the model component provides the business logic. Model components come in many different forms and can be as simple as a basic Java bean or as intricate as Enterprise JavaBeans (EJBs) or Web services.

View Components

View components are used in the MVC architecture to generate the response to the browser. Thus, a view component provides what the user sees. Often times the view components are simple JSPs or HTML pages. However, you can just as easily use WML or another view technology for this part of the architecture. This is one of the main design advantages of MVC. You can use any view technology that you’d like without impacting the Model (or business) layer of your application.

Controller Components

At the core of the MVC architecture are the controller components. The Controller is typically a servlet that receives requests for the application and manages the flow of data between the Model layer and the View layer. Thus, it controls the way that the Model and View layers interact. The Controller often uses helper classes for delegating control over specific requests or processes.

Enter Struts

Although the Model-View-Controller architecture is a powerful means of organizing code, developing such code can be a painstaking process. This is where Struts comes in. Struts is a Web application framework that streamlines the building of Web applications based on the MVC design principles. But what does that mean? Is Struts an MVC Web application that you just add on to or extend? Is Struts just some
Part I: The Struts Framework

libraries? Actually, Struts is a little bit of both. Struts provides the foundation, or framework, for building an MVC-oriented application along with libraries and utilities for making MVC development faster and easier.

You could create a new Controller servlet every time you wanted to use the MVC design pattern in your Web application. Additionally, you’d need to create the management/flow logic for getting data to and from the Model and then routing requests to the View. You’d also need to define interfaces for interacting with your Model objects and all the utility code that goes along with using the MVC design pattern. However, instead of going through this process each time you create a new application, you can use Struts. Struts provides the basic structure and outline for building that application, freeing you to concentrate on building the business logic in the application and not the “plumbing.”

To better understand the benefits of Struts, consider the following analogy. If you were to create a GUI application in Java, you wouldn’t write a textfield widget and a dropdown widget yourself. You would use Java’s Swing API that already has standardized, fully functional code that provides these controls. Not only are the Swing controls ready-to-use, but they are also understood by all Java programmers. Struts provides the same type of advantages: Struts supplies a standard way of implementing an MVC application, the Struts code is tried and true, and the techniques required to use Struts are well known and documented.

In addition to providing the foundation for MVC applications, Struts provides rich extension points so that your application can be customized as you see fit. This extensibility has led to several third-party add-ons being made available for Struts, such as libraries for handling application workflow, libraries for working with view technologies other than JSP, and so on.

Struts Is Open Source

Struts was originally created by Craig R. McClanahan and then donated to the Jakarta project of the Apache Software Foundation (ASF) in 2000. In June of 2001, Struts 1.0 was released. Since then, many people have contributed both source code and documentation to the project and Struts has flourished. Today, Struts has become the de facto standard for building Web applications in Java and has been embraced throughout the Java community. As of the writing of this book, the current version of Struts is 1.2, and Struts is continuing to evolve.

When Craig McClanahan donated Struts to the Apache Jakarta project, it became open source software. This means that anyone can download the source for Struts and modify that code as he or she sees fit. Of course, such changes affect only that developer. The standard code provided by ASF remains unaltered.

Slowly, additional developers were added to the Struts project and were authorized to make changes to the code. These people are known as committers, since they have commit access to the source control repository for Struts. Only ten or so people have
this access, and each picks an area of interest and works on that part of the code that he or she is interested in.

One of the advantages of open source software is that bugs can be fixed in a timely fashion. For ASF projects, bugs are handled by the committers, but anyone can fix a bug and provide a patch that the committers will then evaluate and “commit” if they deem it appropriate. Thus, open source enables rapid development and maintenance cycles. Being open source, Struts is completely free of charge and allows you to make changes to it without any consequence so long as you abide by and preserve the ASF license.

Support for Struts comes in three forms. First is the API documentation that comes with Struts. Second, Struts has a very active mailing list where you can get support for virtually any question. Third, several third-party consulting companies specialize in Struts support and development.

**Basic Components of Struts**

The Struts framework is a rich collection of Java libraries and can be broken down into the following major pieces:

- Base framework
- JSP tag libraries
- Tiles plugin
- Validator plugin

A brief description of each follows.

**Base Framework**

The base framework provides the core MVC functionality and is comprised of the building blocks for your application. At the foundation of the base framework is the Controller servlet: **ActionServlet**. The rest of the base framework is comprised of base classes that your application will extend and several utility classes. Most prominent among the base classes are the **Action** and **ActionForm** classes. These two classes are used extensively in all Struts applications. **Action** classes are used by **ActionServlet** to process specific requests. **ActionForm** classes are used to capture data from HTML forms and to be a conduit of data back to the View layer for page generation.

**JSP Tag Libraries**

Struts comes packaged with several JSP tag libraries for assisting with programming the View logic in JSPs. JSP tag libraries enable JSP authors to use HTML-like tags to represent functionality that is defined by a Java class.
Part I: The Struts Framework

Following is a listing of the libraries and their purpose:

- **HTML**  Used to generate HTML forms that interact with the Struts APIs.
- **Bean**  Used to work with Java bean objects in JSPs, such as accessing bean values.
- **Logic**  Used to cleanly implement simple conditional logic in JSPs.
- **Nested**  Used to allow arbitrary levels of nesting of the HTML, Bean, and Logic tags that otherwise do not work.

**Tiles Plugin**

Struts comes packaged with the Tiles subframework. Tiles is a rich JSP templating framework that facilitates the reuse of presentation (HTML) code. With Tiles, JSP pages can be broken up into individual “tiles” or pieces and then glued together to create one cohesive page. Similar to the design principles that the core Struts framework is built on, Tiles provides excellent reuse of View code. As of Struts 1.1, Tiles is part of and packaged with the core Struts download. Prior to Struts 1.1, Tiles was a third-party add-on, but has since been contributed to the project and is now more tightly integrated.

**Validator Plugin**

Struts comes packaged, as of version 1.1, with the Validator subframework for performing data validation. Validator provides a rich framework for performing data validation on both the server side and client side (browser). Each validation is configured in an outside XML file so that validations can easily be added to and removed from an application declaratively versus being hard-coded into the application. Similar to Tiles, prior to Struts 1.1, Validator was a third-party add-on, but has since been included in the project and is more tightly integrated.

**Acquiring Struts**

Struts is available free of charge and can be downloaded from the Apache Jakarta site at:

http://jakarta.apache.org/struts/

Because Struts is open source, you have a couple of options when downloading the Struts framework software. You can download the software in binary, precompiled form or you can download the source code for compiling on your own. For most cases, the binary distribution will suffice; however, if you want to make changes to the Struts source code, the source distribution is available.

If you choose to download a binary distribution of Struts, you have a couple of options. You can download a released version of the code, which has been rigorously
tested and certified as being of good quality, or you can download a nightly build of the code, which is less stable and not intended for production use. Opting to use a nightly build allows you to get access to the latest enhancements and bug fixes that have been made to the Struts framework ahead of an official release. However, it’s important to point out again that nightly builds have no guarantee on quality because adding a new feature to Struts could potentially break another feature that has been stable for some time.

Similar to downloading a binary distribution of Struts, if you choose to download a source distribution, you have a couple of options. You can download the source for an officially released version of Struts or you can choose to get the “latest and greatest” version of the Struts source code directly from the Struts CVS source control repository. Just as with the binary distribution, choosing to download the latest Struts source code can get you the newest enhancements and bug fixes to the software, but it may also be laden with new bugs.

**What You Get (Binary)**

Since Struts is a Web application framework and not a stand-alone application, Struts distributions are principally comprised of the Struts API libraries and their associated files, such as Document Type Definitions (DTDs) for XML configuration files and JSP Tag Library Descriptor (TLD) files. Additionally, Struts comes with several sample Web applications that illustrate how to use the Struts framework. One of the sample Web applications, `struts-blank.war`, is typically used for new Struts applications because it provides a basic template for a Struts application, including all the necessary .jar files, and so on. Struts distributions also come with a sample Web application, `struts-example.war`, that illustrates the basic structure of a Struts application.

**What You Get (Source)**

Similar to the binary distribution, the source distribution is comprised of the Struts API libraries and sample Web applications. The major difference, however, is that all of the code for the libraries and sample applications is in source form. This is particularly useful for projects where the source code may need to be changed or where you may want access to the source code for debugging an application and so on.

**Getting Started with Struts**

Now that the theoretical foundation for Struts has been covered, it is time to move on to actually writing Struts code. The next chapter walks through an example Struts application. Before then, you will need to choose one of the two Struts distribution options just discussed and download it.